Extracting Kozak Consensus Sequence Using Kleisli

Jing Chen* Nam-Hai Chua† Daphna Strauss‡ Limsoon Wong§¶

29 April 1998

Abstract

Consensus sequence for the context of translation initiation codon is useful to many molecular biologists in identifying the possible translation initiation codon in new genes. We have two objectives in this paper. First, we want to build a system that can extract context sequences of translation initiation codon from large DNA databases and compute a consensus sequence for these context 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 such a system by integrating WU-BLAST2.0, Entrez, and database-style indices.

1 Introduction

The recent explosion of genomic information, as gleaned from the Human Genome Project and other similar efforts, has been fueled by engineering and technological advances. However, as the amount of information grows the challenge becomes one of managing and making sense of the data. In fact, biological discovery is increasingly performed in a "dry and wet" paradigm, in which hypotheses are first formulated by analyzing existing data sources and subsequently confirmed by experiments. This paradigm relies on the development of a number of tools that require advances in the fundamentals of computer science as well as careful engineering. "Dry" experiments inevitably involve integrating and transforming data souces so that they can be analyzed or "mined" with the appropriate software. Integration of these data sources is complicated by the fact that they use different formats and require different languages for accessing them. One of the first challenges that must be addressed therefore is to provide the appropriate tools to scale this "tower of babel." [1]

Kleisli [?, ?] 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

*Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613. Email: cjing@krdl.org.sg.
†The Rockefeller University, 1230 York Avenue, New York, NY 10021-6399. Email: chua@rockvax.rockefeller.edu.
‡Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613. Email: daphna@krdl.org.sg.
§Contact author.
¶Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613. Email: limsoon@krdl.org.sg.
[1]Taken from Susan Davidson's talk on 28 April 1998 at a University of Pennsylvania's IME seminar.

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 [**?**, **?**] 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.

We demonstrate here the use of Kleisli to solve an interesting problem in bioinformatics: the extraction of Kozak sequences from large genomic databases and the derivation of a consensus for them. The context of the translation initiation codon in genomic sequences is called Kozak sequence [**?**]. Ten years ago, Joshi [**?**] proposed a consensus Kozak sequence for the translation initiation codon in higher plants on the basis of 79 genomic sequences. His result was useful to many plant researchers in identifying the possible translation initiation sites in new genes. The distribution of sequences in genomic databases such as GenBank [**?**] is skewed, with certain gene families heavily over represented. Fortunately, the size of GenBank is increasing rapidly. As the size increases, the coverage of GenBank of gene families becomes more extensive and less biased. Therefore, it is worthwhile to periodically extract Kozak sequences for various organisms and compute their consensus. Also, the coverage of GenBank on certain organism is now large enough that it is feasible to consider extracting Kozak sequences and computing their consensus with respect to functional classification of genes within one species or across entire family of species. The problem of extracting Kozak sequences and computing their consensus with respect to functional classification of genes is quite typical of many bioinformatics problems. We use for illustration here the energy-related genes of the model dicot *Arabidopsis*, whose entire genome is currently being sequenced [**?**].

On 25 April 1998, one of us used the specification "energy AND arabidopsis[Organism]" to search the GenBank portion of the popular Entrez system [**?**] at the National Center for Biotechnology Information (NCBI) in Washington DC to get these "energetic" Arabidopsis genomic sequences. One needed to get hold of them before one could proceed to extract and analyse their Kozak sequences. It returned a list of exactly one record. There were actually more "energetic" Arabidopsis genes than one in GenBank. There are two reasons associated with this poor recall performance. The first is that the specification is slightly simple minded. The second is that records in GenBank are poorly annotated and some are even unannotated. The first problem can be rectified by a more sophisticated specification that more fully exploit the biological meaning of "energy-related." However, no amount of fixing of the Entrez search specification can deal with the second factor of poor recall performance. So we can see that without the support of good tools, it is not even possible to start!

Let us try to describe what kind of tools are needed. The most straightforward way to find most of the energy-related Arabidopsis sequences is to rely on the fact that sequences having a high degree of homology are likely to have highly similar functions. So one should ask Entrez for all energy-related sequences, regardless of species. Even though some specific energy-related sequences of one species might be missed out, hopefully a similar one from another species would be picked out to fill the gap. Then we can compare all Arabidopsis sequences to this set of known energy-related sequences from other species. Those that exhibit strong homology would be the energy-related Arabidopsis sequences. In order to carry out this procedure smoothly, one has to integrate Entrez with a reliable homology search system such as WU-BLAST2.0 [**?**, **?**]. The procedure as described above can identify energy-related Arabidopsis sequences. However, we are still a little far from being able to extract their Kozak sequences. To do that we need to first locate the start codon of each sequence. To locate the start codon we need to inspect the so-called feature table of each sequence record. The feature table gives the positions of various features in a sequence, including positions of its first exon and thus the location of its translation start codon. However, feature tables in GenBank reports are considerably more complicated than simple flat relational tables. So we would also need a system that can manipulate these tables with ease. Assuming that we can manipulate feature tables and thus extract Kozak sequences, we would still need a way to analyse them in order to calculate their consensus using, for example, the 50/75 consensus rule described by Cavener[**?**].

The modern database integration system Kleisli possesses all the necessary attributes to address the requirements above. This article focuses on showing exactly how Kleisli can be used to extract and analyse Kozak sequences from Entrez. We organize our presentation into the following sections. Section 2 is a quick introduction to the complex object data model of Kleisli and to its high-level query language called CPL (short for Collection Programming Language). In particular, we focus on its ability to manipulate feature tables from GenBank reports. Section 3 describes the programming of the basic steps of extracting Kozak sequences of all Arabidopsis sequences from the GenBank portion of Entrez. Section 4 presents the calculation of consensus Kozak sequence using the popular 50/75 rule [**?**]. Having established the basics, we proceed to Section 5 which shows we can integrate Entrez and WU-BLAST2.0 to extract and analyse Kozak sequences with respect to functional classification of sequences.

2 A 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 some integrated version of the available data. In this quick tour, we focus on the data model of Kleisli that is based on complex object types and on the high-level query language supported by Kleisli called CPL, which stands 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 [**?**] 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 will not be using variants here.

As an example of a complex type, consider the following type that represents the feature table of a GenBank report from Entrez [**?**].

```
(#uid:       num,
 #title:     string,
 #accession: string,
 #feature: { (#name:  string,
              #continuous: bool,
              #position: [ (#accn: string,
                            #start: num,
                            #end:   num,
                            #negative: bool,
                            #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 GenBank report. The `#feature` field is a set of records, which together form the feature table of the corresponding GenBank report. Each of these records has three fields `#name`, `#continuous`, and `#position`. The `#position` field is itself a list of position records giving positions of the corresponding feature. A position record is quite complicated and deserves a more detailed explanation. A feature in a DNA sequence is generally made up of exons, which are segments of the DNA sequence that actually code for proteins. In a DNA sequence, exons are separated by noncoding regions called introns. The DNA sequence can also either be on the positive or the negative strand. Each position record of a feature corresponds to an exon of that feature. If the DNA sequence is from the positive strand, then its exons are listed from the 5'-end to the 3'-end of the DNA sequence; the 5'-end corresponds to position 0 of the DNA sequence, the 3'-end corresponds to the last position of the sequence. If the DNA sequence is from the negative strand, then its exons are listed the other way round. Due to the nature of sequencing experiments, the pieces of a DNA sequence are often obtained at different times. Occasionally, some pieces are still missing at assembly time. The `#continuous` field of indicates if all the pieces constituting the feature are present. The `#start` and `#end` fields of a position record correspond, respectively, to the start and end positions of the corresponding exon. The `#negative` field is used to indicate if the exon is on the negative strand. If the `#negative` field is true, then the exon is on the negative strand; it must be reverse-complemented according to the famous Crick-Watson rule [**?**] to obtain the corresponding sequence on the positive strand. Each exon of a feature can be annotated; the `#anno` field of the position record stores the annotations. Lastly, the DNA sequences can be assembled from some previously recorded sequences, some exons from its features can come from these prevously recorded sequences. The `#accn` field of each position record stores the accession number of the DNA sequence from which the corresponding exon is obtained.

In general, the types are given by the syntax: t ::= `num` | `string` | `bool` | {t} | {|t|} | [t] | (l_1 : t_1, ..., l_n : t_n) | <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: 199964,
  #accession: "M12345",
  #title: "Mouse (ST4) c-myc proto-oncogene, promoter region.",
  #feature: {(#name: "source",
              #continuous: true,
              #position: [(#accn: "M12345",
                           #start: 0,
                           #end: 1539,
                           #negative: false)],
              #anno: [(#anno_name: "organism",
                       #descr: "Mus musculus"),
                      (#anno_name: "db_xref",
                       #descr: "taxon:10090")]),
...})
```

This is a rich data model. As can be seen from the complicated GenBank report, schemas and structure of all popular bioinformatics databases, flat files, and softwares are easily mapped into it. `//adenine.krdl.org.sg:8080/publications/drivers.html` is a list of external sources that Kleisli talks to.

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 [**?**], we illustrate it through a series of examples.

The first example extracts the titles and features of a set `DB` of feature tables. 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`.

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

The second example extracts the titles and features of those elements of `DB` whose titles contain `tyrosine` as a substring.

```
{ (#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.

```
{ (#title: x.#title, #feature: f.#name,
   #position: f.#position, #annotations: f.#anno)
| \x <- DB,
  \f <- x.#feature };

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

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

let \DB' == { (#entry: x, #organism: a.#descr)
            | \x <- DB, \f <- x.#feature,
              \a <--- f.#anno, a.#anno_name = "organism" } in
let \ORG == { y.#organism | \y <- DB' }
in [ (#organism: z,
      #entries: { v.#entry | \v <- DB', v.#organism = z })
   | \z <--- [ u | \u <- ORG ] ];
```

The first query 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. The second query flattens `DB` completely. The `\a <--- f.#anno` has similar meaning to `\x <- DB`, but works on lists instead of set. Thus it binds `a` to each item in the list `f.#anno`. Similarly, `p` is bound to each item in the list `f.#position`. The third query 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. The fourth query demonstrates how to do nesting in CPL. The subquery `DB'` is the restructuring of `DB` as in the third query described above. The subquery `ORG` then extracts all organism names. The main query groups entries in `DB'` by organism names. It also sorts the output list by alphabetical order of organism names, because `[ u | \u <- ORG]` converts the set `ORG` 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 [?, ?, ?, ?] and references thereof.

3 Extracting Kozak Sequences

In this section we explain the basic steps of extracting Kozak sequences of the model dicot *Arabidopsis*. We use the GenBank section of Entrez at Washington DC to be our source of genomic sequences.

Step 1. It is only possible to extract Kozak sequences from genomic sequences whose coding regions are explicitly annotated. The Entrez specification to download Arabidopsis DNA sequences having coding regions that are explicitly annotated is "Arabidopsis[Organism] AND cds[Feature key]." The CPL function `na-get-uid-general` remotely queries Entrez for unique identifiers of GenBank reports satisfying a given specification.[2] Thus to find out which Arabidopsis sequences have annotated coding regions, we just apply `na-get-uid-general` to the specification "Arabidopsis[Organism] AND cds[Feature+key]" and write the unique identifiers obtained to a file `uids` for later use, as shown in the CPL program below.

```
writefile na-get-uid-general ("Arabidopsis[Organism]+AND+cds[Feature+key]")
to "uids" using stdout;
readfile uids from "uids" using stdin;
```

Step 2. Our second step is to obtain all Arabidopsis DNA sequences whose coding regions are explicitly annotated. The CPL function `na-get-seq-by-uid` downloads from Entrez the DNA sequence uniquely identified by its input argument. So for each unique identifier `u` in `uids`, we apply `na-get-seq-by-uid` to `u` to get the corresponding sequence `s`. These sequences are written to the file `seq`. A database-style index `accn2seq` is then created on the `#accession` field of the sequence file `seq`, so that we have fast access to a sequence given its accession number. The CPL program implementing all these is given below.

```
writefile
  { s | \u <- uids,
        \s <- na-get-seq-by-uid (u) }
to "seq" using stdout;
writefile "seq" to "#accession" using setindex-create;
setindex-access (#name: "accn2seq", #file: "seq", #key: "#accession");
```

Step 3. Before we proceed to obtain the feature table of each sequence, we need several functions to implement the extraction of a Kozak sequence given position information. There are two functions, `get-+ve-strand` for extracting from the positive strand and `get--ve-strand` for extracting from the negative strand. These functions are implemented as shown below.

```
primitive get-+ve-strand == \p =>
  { left ^ "ATG" ^ right
  | (#sequence: \s, ...) <- process <#key: p.#accn> using accn2seq,
    \exon == string-span (s, p.#start, p.#end),
```

[2] The details of `na-get-uid-general` and other functions of CPL can be found in [?].

```
    exon string-islike "ATG%",
    \left == string-span (s, p.#start - 15, p.#start - 1),
    string-length (left) = 15,
    \aug == string-span (s, p.#start, p.#start + 2),
    \right == string-span (s, p.#start + 3, p.#start + 12),
    string-length (right) = 10 };

primitive get--ve-strand == \p =>
  let \complement == \s => string-rev (string-trans ("ACGT", "TGCA", s))
  in { left ^  "ATG" ^ right
     | (#sequence: \c, ...) <- process <#key: p.#accn> using accn2seq,
       \s == complement (c),
       \b == string-length (c) - (p.#end + 1),
       \e ==  string-length (c) - (p.#start + 1),
       \exon == string-span (s, b, e),
       exon string-islike "ATG%",
       \left == string-span (s, b - 15, b - 1),
       string-length (left) = 15,
       \right == string-span (s, b + 3, b + 12),
       string-length (right) = 10 };
```

We describe `get-+ve-strand` now. Its input `p` is a position record, already described in Section 2, specifying how the first exon is derived. The `#accn` field of `p` gives the accession number of the sequence of this exon. We fetch the sequence `s` from our index `accn2seq`. Then we use the `#start` and `end` fields of `p` to extract from `s` the specific segment `exon` constituting the exon. We make sure that `exon` is indeed the first exon by testing that t begins with `ATG`. The upstream component of the Kozak sequence `left` is extracted from `s`. We test to make sure that it has exactly 15 nucleotides (15 bases). The downstream component of the Kozak sequence `right` is extracted from `s`. We test to make sure that it is 10 nucleotides long. The Kozak sequence is then built by concatenating `left`, `ATG`, and `right`. The `get--ve-strand` function is similar, except we need to reverse-complement the exon.

Step 4. We can now proceed to get our Kozak sequences. For each unique identifiers `u` in `uids`, we apply the CPL function `na-get-feature-by-uid` to fetch its feature table `x` from Entrez. We iterate through each feature `f` in `x` to look for feature that corresponds to coding regions; these are those feature whose name is `CDS`. We have to make sure that the feature is complete, hence the test `f.#continuous`. To further make sure that the coding region is proper, we check to see if its annotations `a` contain a protein translation and if that translation begins with methionine (`M`). If all these tests pass, we set `p` to be the position record of the first exon of this feature. Then depending on whether the exon is on the positive or negative strand, we invoke `get-+ve-strand` or `get--ve-strand` to get the corresponding Kozak sequence `z`. The Kozak sequences, the unique identifiers, and the translations are all written to the file `kozak` for subsequent processing.

```
writefile
  { (#uid: x.#uid,
     #translation: a.#descr,
```

```
     #kozak: z)
   | \u <- uids,
     \x <- na-get-feature-by-uid (u),
     \f <- x.#feature,
     f.#continuous,
     f.#name = "CDS",
     \a <--- f.#anno,
     a.#anno_name = "translation",
     a.#descr string-islike "M%",
     \p == f.#position.list-head,
     \z <- if p.#negative then get--ve-strand (p) else get-+ve-strand (p) }
to "kozak" using stdout;
readfile kozak from "kozak" using stdin;
```

4 Computing Consensus

All the Kozak sequences of Arabidopsis are in the set `kozak`. We can now proceed to calculate a consensus sequence for them. We determine the consensus sequence using the 50/75 rule described by Cavener [**?**] and used in many other papers [**?**, **?**, **?**, etc.]

Step 5. The consensus at a position is computed according to the following rules, with decreasing order of priority. (1) If a nucleotide at that position has a relative frequency greater than 50% and greater than twice the relative frequency of the second most frequent nucleotide, the nucleotide is given consensus status and is indicated in uppercase. (2) If the sum of the relative frequencies of a pair of nucleotides exceeds 75%, these two nucleotides are given co-consensus status are are indicated in uppercase. (3) If there is a single most frequent nucleotide, it is given dominant status and is indicated in lowercase. (4) If two bases have the same highest frequency, they are given co-dominant status and are indicated in lowercase. This calculation is defined in the CPL function below.

```
primitive mk-consensus == \C =>
  (list-gensort @ ((\x, \y) => x.#c < y.#c) @ C) before (\C' =>
  let \_1st == C'.list-head in
  let \_2nd == C'.list-tail.list-head in
  let \_3rd == C'.list-tail.list-tail.list-head in
  let \_4th == C'.list-tail.list-tail.list-tail.list-head
  in if (_1st.#c > 50) andalso (_1st.#c > (2 * _2nd.#c))
     then [ string-toupper (_1st.#l)]
     else if (_1st.#c + _2nd.#c) > 75
     then [ string-toupper (_1st.#l), string-toupper(_2nd.#l)]
     else if _1st.#c > _2nd.#c
     then [ string-tolower (_1st.#l)]
     else [ string-tolower (_1st.#l), string-tolower (_2nd.#l)]);
```

Here C is the input which represents the relative frequencies of nucleotide letters at a certain position. It is just a list of four records. Each record has a #l field which stores a nucleotide letter and a #c field which stores the relative frequency of that letter. The list C is then sorted with respect to the relative frequencies of the four nucleotide letters using the list-gensort function of CPL. Then 1st, 2nd, 3rd, and 4th is assigned respectively the most frequent, second most frequent, third most frequent, and fourth most frequent nucleotide letters and their frequencies. The if-then-else block simply performs the 50/75 calculation as described.

Step 6. Next we apply mk-consensus to each position. The CPL program implementing this step is given below.

```
primitive generate-consensus == \kozak =>
  let \nr == [ k | \k <- { x.#kozak | \x <- kozak }] in
  let \mk-pc == \c => ((100 * c / list-numitems (nr)) + 0.5) mod 101 in
  let \hist == list-irs
         @ ((\s, \S) => [ if string-substring (s, c.#p, 1) = "A"
                          then modify #a of c to 1 + c.#a
                          else if string-substring (s, c.#p, 1) = "C"
                          then modify #c of c to 1 + c.#c
                          else if string-substring (s, c.#p, 1) = "G"
                          then modify #g of c to 1 + c.#g
                          else modify #t of c to 1 + c.#t
                        | \c <--- S ],
            [ (#p: p, #a: 0, #c: 0, #g: 0, #t: 0)
            | \p <--- [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, 26, 27 ]])
         @ nr
  in [ (#position: p, #consensus: mk-consensus (C),
        #a-pc: a-pc, #c-pc: c-pc, #g-pc: g-pc, #t-pc: t-pc)
     | \q <--- hist,
       \a-pc == mk-pc (q.#a), \c-pc == mk-pc (q.#c),
       \g-pc == mk-pc (q.#g), \t-pc == mk-pc (q.#t),
       \C == [ (#l: "a", #c: a-pc), (#l: "c", #c: c-pc),
               (#l: "g", #c: g-pc), (#l: "t", #c: t-pc) ],
       \p == if q.#p < 15 then num-stringify (q.#p - 15)
             else if q.#p = 15 then "A"
             else if q.#p = 16 then "T"
             else if q.#p = 17 then "G"
             else num-stringify (q.#p - 14) ];
```

Here are the details. The Kozak sequences are sorted and duplicates are eliminated from them, because they represent identical sequences from two or more laboratories. The sorted duplicate-free list of Kozak sequences is stored in nr. To produce a more readable frequency table, we also define a function mk-pc to convert the count of a nucleotide letter into relative frequency, rounded to two decimal point. Next we use the `list-irs` structure recursion operation of CPL to generate a count of each nucleotide at each position. Structural recursion satisfies the equation $\texttt{list-irs}(i, e)[o_1, \ldots, o_n] = (\cdots((e\ i\ o_1)\ i\ o_2)\ i\ \cdots\ i\ o_n)$, see [?]. Let us use the equation above to explain what ends up in `hist`. The o_i are Kozak sequences in `nr`. The `e` is simply the initial frequency list of twenty-eight records storing the initially frequencies (= 0) of each of the four nucleotide letters `a`, `c`, `g`, and `t` respectively in fields `#a`, `#c`, `#g`, and `#t` at each position indicated in the `#p` field. The i is a function that given a Kozak sequence `s` updates the accumulated frequency list `S` accordingly. Thus `hist` is just the final frequency list or more prosaicly, a histogram of the occurrences of nucleotides. The histogram is then processed to give the consensus. Basically, for each frequency record `q` of the histogram, we apply `mk-pc` to obtain the relative frequencies `a-pc`, `c-pc`, `g-pc`, and `t-pc` of the four nucleotides at the position. The relative frequencies are stored into the list `C`. The function `mk-consensus` defined in Step 5 is then applied to `C` to obtained the consensus at this position. Since it is desirable to print the position using "biologist" notations, the `if-then-else` block defining `p` is doing precisely this.

Step 7. Having developed the `generate-consensus` function, all we need to do to get the consensus Kozak sequence for Arabidopsis is to run it on the set `kozak`!

```
generate-consensus (kozak);
```

For completeness, the output from the above is shown in the two tables below. The first table contains the context information up to 15 bases upstream from the translation initiation codon ATG. The second table contains the context information up to 10 bases downstream of ATG. The consensus calculated is

```
(t/a)aaaaaaaaaaAaaATGGc(t/g)actaata
```

It is consistent with the consensus

```
aaaaaaaA(A/C)aATGGCtacta(c/t)ta
```

on dicots in general [?]; recall that Arabidopsis belongs to the Dicotyledon family! For example, the context upstream of ATG in dicots is predominantly `a` and `t`, as is our case here in Arabidopsis; and the -3 position upstream of ATG in dicots is `A`, as is our case here in Arabidopsis. However, there are also some differences between Arabidopsis and dicots in general. For example, at the +5 position downstream from ATG, the consensus for dicots is `C` at 60% frequencies [?], but it is only a `c` at 40% frequency for Arabidopsis.

position	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
A-%	31	32	32	33	35	35	32	34	35	35	32	44	51	43	44
C-%	18	20	19	18	18	17	18	22	17	16	25	14	10	30	20
G-%	19	18	19	19	19	21	22	17	21	23	17	23	25	9	24
T-%	31	29	30	30	28	28	28	27	28	26	26	19	14	18	12
consensus	ta	a	a	a	a	a	a	a	a	a	a	a	A	a	a

position	A	T	G	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13
A-%	100	0	0	20	30	22	34	30	27	32	29	28	31
C-%	0	0	0	7	40	12	16	31	17	18	28	19	17
G-%	0	0	100	58	15	33	29	16	25	28	15	23	30
T-%	0	100	0	15	15	33	22	23	31	23	27	31	21
consensus	A	T	G	G	c	tg	a	c	t	a	a	t	a

As can be seen, relatively simple and short CPL programs can be written and executed in Kleisli to extract and generate consensus of context sequences of translation initiation sites in DNA sequences.

5 Classification by Function

Genes of an organism have different functions and fall into general function classes such as energy, signal transduction, metabolism, and so on [?]. Having a good idea of the context sequences of translation initiation codon may be useful in understanding the function of novel genes. Now that we have gone through the basic steps of extracting and analysing Kozak sequences for an entire organism, let us discuss the detail of extracting and analysing Kozak sequences with respect to such function classes. Since we have already downloaded the sequences of Arabidopsis, we continue with them. As our example, we want to extract Kozak sequences from Arabidopsis sequences involved with energy-related functions. Here are the steps.

Step 8. As discussed earlier, we need to first use known energy-related sequences from other organisms to identify energy-related Arabidopsis sequences in our collection. We accomplish this by downloading all energy-related protein sequences from Entrez and constructing a BLAST database on them. The CPL script for this step is shown below.

```
writefile
  { (#uid: x.#uid,
     #accession: x.#accession,
     #title: x.#title,
     #seq: x.#sequence)
  | \u <- aa-get-uid-general "energy",
    \x <- aa-get-seq-by-uid (u), x.#uid = u }
to "energy-blast" using localblast-setdb;
localblast-blastp (#name: "energy-blast", #db: "energy-blast");
```

The `aa-get-uid-general` function of CPL is used to obtained unique identifiers `u` of protein sequences in Entrez. The Entrez search specification used here is the string `"energy"`. It can be easily replaced by a more refined specification; but this is outside the scope of this paper. The protein sequence `x` of `u` is then downloaded from Entrez using the CPL function `aa-get-seq-by-uid`. The downloaded sequences are written to the file `energy-blast` as a BLAST database using the `localblast-setdb` function. A BLASTP connection `energy-blast` is open to this file using the `localblast-blastp` function. Subsequently, we can perform BLASTP on a sequence against this database of energy-related sequences simply by processing the sequence using this connection.

Step 9. We can now identify which Kozak sequences in the file `kozak` obtained in Step 4 are from enery-related Arabidopsis genes. Recall that each record in `Kozak` has a `#translation` field. This field stores the protein sequence of the corresponding genes. So all we have to do now is to process this protein sequence using BLASTP to see if it has significant homology to some of the energy-related sequences. The CPL program is shown below.

```
writefile
  { x | \x <- kozak,
        set-thereis
          @ (\h => h.#pscore < 1.0E~6)
          @ (process x.#translation using energy-blast) }
to "energy-kozak" using stdout;
readfile energy-kozak from "energy-kozak" using stdin;
```

Here we iterate the following for each `x` in `kozak`. BLASTP is performed on the protein translation of `x`. We then check if there is a hit `h` whose `pscore` is significant. We use 0.000001 as our threshold here. If a significant hit is found, we include `x` in the output, which is written to the file `energy-kozak`.

Step 10. The Kozak sequences of all energy-related Arabidopsis sequences are now stored in the set `energy-kozak`. All that remain to calculate their consensus is to invoke the `generate-consensus` function developed at Step 6 on this set.

```
generate-consensus (energy-kozak);
```

This completes our implementation. We use Arabidopsis and energy-related sequences as our examples, our CPL programs can be copied verbatim, modulo minor changes in the Entrez search specifications, to extract and analyse Kozak sequences with respect to other organisms and function classes.

6 Conclusion

We demonstrated two things in the previous sections. First, we constructed a useful system that can extract and analyse Kozak sequences in a powerful and precise way from large genomic databases like GenBank. In particular, we can easily produce consensus Kozak sequence with respect to functional classification of genes. Second, we illustrated the ease and clarity that such a useful integrated system can be constructed in the modern database integration system Kleisli using its high-level query language CPL.

As can be seen from the foregoing, it is quite easy to solve this and other kinds of bioinformatics problems involving integration of unusual databases and analysis tools using CPL in Kleisli. We also believe that the Kleisli/CPL technology is applicable in more general database integration problems.

There are two main directions to proceed with respect to Kozak sequences. The first is to make use of our system to perform a comparative study of context sequences of translation initiation codon across different organisms and across different function classes within one organism. Such a study should be repeated at suitable interval because the size of genomic databases increases with respect to time and it is of interest to examine if conclusions based on current data will still be valid in the presence of more extensive data in future. Some preliminary results along this direction have already been obtained and are available at `//adenine.krdl.org.sg:8080/~limsoon/kozak`.

The second direction is to make use of the Kozak sequences and consensus obtained to develop reliable systems to predict translation initiation site of DNA sequences from various organisms and function classes. The identification of translation initiation site is an important step in discovering new genes. Some systems already exist for predicting translation initiation sites. For example, in a 1997 study involving about 500 Arabidopsis sequences, NetStart [**?**] was able to correctly identify nearly 90% of translation initiation codons with negligible false positives. We should test it again using our considerably larger number of Arabidopsis sequences. If the test is promising, we should then use our Kozak sequences to train neural networks similar to NetStart to predict translation initiation sites of other species (especially amongst plants) and of selected function classes.