MSF to FASTA Conversion Using Kleisli

Limsoon Wong*

8 May 1998

1 Problem

GCG uses a file format called MSF. Another popular format used by molecular biologists is FASTA. An example of the former is given in Figure 1. an example of the latter is given in Figure 2.

An MSF file has two parts, the part above the `//` is the header information and the part below the `//` is the alignment information. We want to extract the protein sequences from the alignment information part of an MSF file and put these sequences into FASTA format.

Why are we doing this? There are two reasons. The first is that the popular `READSEQ` program in the web[1] does not perform this conversion correctly. The second is that this conversion is a good exercise on some of the string manipulation capability in our Kleisli/CPL system[1].

2 Solution

The Kleisli/CPL program that converts MSF multiple sequence alignment file into FASTA format is given in Figure 3. The line numbers are given for ease of reference in my explanation below. It handles the desired MSF-FASTA conversion. It also works when the header information part of an MSF file is missing. You just substitute `MY-MSF-FILE` by the name of your input MSF file and `MY-FASTA-FILE` by the name of the desired output FASTA file. Then submit it to the Kleisli/CPL system for execution. Easy!

3 Explanation

Okay, let us go over the program in Figure 3 so that you know how it works.

*Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, singapore 119613. Email: limsoon@krdl.org.sg

[1]See `http://bimas.dcrt.nih.gov/molbio/readseq/`.

```
!!AA_MULTIPLE_ALIGNMENT 1.0
PileUp of: *.sw

 Symbol comparison table: GenRunData:blosum62.cmp  CompCheck: 6430

                   GapWeight: 12
             GapLengthWeight: 4

 hox1.msf  MSF: 342  Type: P  February 25, 1998 11:37  Check: 5044 ..

 Name: hxd1_mouse       Len:   342  Check: 6864  Weight:  1.00
 Name: hxd1_xenla       Len:   342  Check: 5615  Weight:  1.00
 Name: hxa1_human       Len:   342  Check:  122  Weight:  1.00
 Name: hxa1_mouse       Len:   342  Check: 9465  Weight:  1.00
 Name: hxa1_xenla       Len:   342  Check: 2241  Weight:  1.00
 Name: hxb1_human       Len:   342  Check: 3301  Weight:  1.00
...

//

            1                                                    50
hxd1_mouse  ~~~~~MSSYL EYVSCAAGGG SGGVGGDVLA FAPKFC.ADA RPVALQPAFP
hxd1_xenla  ~~~~~MNSYL EYTSC..... .....GDVLA FSPKFCRSDH RNMNLQP.YP
hxa1_human  ~~~~~MNSFL EYPILSSGDS GTCSARAYPS DHRITTFQSC AVSANSCGGD
hxa1_mouse  ~~~~~MNSFL EYPILGSGDS GTCSARAYPS DHGITTFQSC AVSANSCGGD
hxa1_xenla  ~~~~~~~~~~ ~~~~~~~~~~ ~~~~~~~~~~ ~~~~~~~~~~ ~~~~~~~~~~
hxb1_human  MDYNRMNSFL EYPLCNRG.P SAY.S....A HSAPTSFPPS S.AQAVDSYA
hxb1_mouse  MDYNRMSSFL EYPLCNRG.P SAY....... .SAPTSFPPC S.APAVDSYA
hxb1_chick  MDNTRMNSFL EYAICNRG.T GAYHH....L EQSSPSFPSC SGSPSSDTFH
hxb1_ambme  ~~~~~~~~~~ ~~~~~~~~~~ ~~~~~~~~~~ ~~~~~~~~~~ ~~~~~~~~~~
hxa1_brare  ~~~~~MNSYL DYTIYNRG.S NTYSSKVGCF PVEQEYLP.. SACASTNSYI
hxb1_cypca  MDNSSMNSFL EYTICNRG.T NAYSPKAGYQ HLDQALSGPF HTGHARYSHN

            51                                                   100
hxd1_mouse  LGNGDGAFVS CLPLATARPT PSPPAGPAQS PVPQPAAPRY APCTLEGAYE
hxd1_xenla  RSGADHPFM. ........PV GGVP.GSVAH QAPHQSPALY APCSLDVAYE
...
```

Figure 1: An example MSF file.

```
>hxd1_mouse
MSSYLEYVSCAAGGGSGGVGGDVLAFAPKFCADARPVALQPAFPLGNGDG
AFVSCLPLATARPTPSPPAGPAQSPVPQPAAPRYAPCTLEGAYERGAAPA
SAAEYGFLGSGPAFDFPGALGRAADEGGAHVHYATSAVFSGGGSFLLSGQ
VDFAAFGEPGPFPACLKEPADGHPGPFQTVSPAPGACPKPASPTSSLPAA
HSTFEWMKVKRNAPKKSKLSEYGATSP
>hxd1_xenla
MNSYLEYTSCGDVLAFSPKFCRSDHRNMNLQPYPRSGADHPFMPVGGVPG
SVAHQAPHQSPALYAPCSLDVAYEPTPPSDYSFLHSSPDYDYYGSNHQIE
ETGGHIPYGSSLFPGNGSYILNGQLSYRTLGEETQMAQCKEPLEVYPGGN
LQSISPSPGTYPKPASPASDSYVSTFDWMKVKRNPPKKSIPSEYGVTSP
...
```

Figure 2: An example FASTA file.

```
primitive raw == list-irs
  @ ((\x, \y) => case y
       of <#skip: _> => if x string-islike "//%" then <#noskip: []> else y
        or <#noskip: \z> => <#noskip: z [+] [ x]>,
    <#skip: ()>)
  @ process "MY-MSF-FILE" using file-intolines ;

writefile
  [ (#title: t, #seq: s)
  | \x <--- case raw
             of <#noskip: \r> => r
             or <#skip: _> => process "MY-MSF-FILE" using file-intolines,
    not (x = ""), not (x string-islike " %"),
    \t == string-span (x, 0, 9),
    \s == string-implode (string-tokenize (" ~.", string-suffix(x,10), 0))]
to "/tmp/fragments" using stdout;

writefile "/tmp/fragments" to "#title" using listindex-create;
listindex-access (#name:"name2seq", #file:"/tmp/fragments", #key:"#title");

writefile
 { (#title: x, #seq: S)
 | \x <- set-unique{ x.#title | \x <--- process <#all: ()> using name2seq},
   \S == string-implode[ y.#seq | \y <--- process <#key: x> using name2seq]}
to "MY-FASTA-FILE" using fasta-out;
```

Figure 3: The Kleisli/CPL program that converts MSF to FASTA format.

Lines 1–6 define `raw` to be the variant `<#skip:()>` if your input MSF file does not have the header information part. On the other hand, if it does have the header information part, then `raw` is defined to be the variant `<#noskip:` r`>`, where r is a list of lines from the multiple alignment part of your MSF file. This is perhaps the most interesting part of the program. I will say more about it shortly.

Lines 7–15 try to break the multiple alignment part into a list of records having a title field and a sequence field. In line 9, we test `raw` to see which of the two variant form it is. Line 10 is for the case when it is `<#noskip:` r`>`; we let the variable `x` range over the lines in r. Line 11 is for the case when it is `<#skip:()>`; we know that the MSF file lacks header information, so we let `x` range over all the lines in the MSF file. Line 12 removes every `x` that corresponds to an empty line or to a position indicator line of the alignment partof the MSF file. Line 13 extracts the title from `x`; the title is just the first 10 letters. Line 14 extracts the sequence segment from `x`, deleting space and insert-delete symbols. The resulting list is written to a temporary file.

Lines 16–17 create an index on the title field of this temporary file.

Lines 18–22 uses this index to group sequence segments having the same title. The sequences in each group are then "imploded" or concatenated together to recover the complete sequence. The recovered sequences are then written to the desired FASTA file in FASTA format.

Let us return to the definition of `raw`. The function CPL `list-irs` obeys this equation:

$$\texttt{list-irs}(f, e)[o_1, \ldots, o_n] = e\ f\ o_1\ f\ \cdots\ f\ o_n$$

with f written in infix form and associating to the left. From line 2, we see that `x` is set to o_i at the ith application of f and y is set to the result of the $(i-1)$th application of f. From line 5, we see that the e in the equation above is "initialized" to `<#skip:()>`. So initially, y is `<#skip:()>`. From line 3, we see that y remains as `<#skip:()>`, until we see the first occurrence of `//`. At which point, y takes on the form `<#noskip:` z`>` where z are the lines we have seen so far after the `//`. Therefore, lines 2–5 have the overall effect of skipping everything until we see the first `//` and then returning everything else.

That's all folks. If you want more detail on the syntax and available primitives in Kleisli/CPL, please refer to the manual [2].

References

[1] Susan Davidson, Christian Overton, Val Tannen, and Limsoon Wong. BioKleisli: A digital library for biomedical researchers. *International Journal of Digital Libraries*, 1(1):36–53, April 1997.

[2] Limsoon Wong. *The Collection Programming Language Reference Manual.* Institute of Systems Science, Heng Mui Keng Terrace, Singapore 119597, October 1995. Available at `http://sdmc.iss.nus.sg/kleisli/psZ/cpl-defn.ps`.