Student Project :

Designing an Organisms-Genomes Database written in *OCaml*

Y. Pageaud

Institute of Biology, Genetics and BioInformatics (IBGBI), 91000 Evry, France

December 14, 2014

ABSTRACT

The idea was to design a pseudo database to link data on organisms and data on their genomes, all operating with recursive functions. Queries are also recursive functions. Here is a way to do it.
NB : some of the keywords used in the script are in french.

We create the record type taxonomie_t. It will contain all taxonomies :

```
# type taxonomie_t = Bacterie|Eucaryote|Archee;;
type taxonomie_t = Bacterie | Eucaryote | Archee
```

We create the record type ecotype_t. It will contain all ecotypes :

```
# type ecotype_t = Mesophile|Thermophile|Psychrophile;;
type ecotype_t = Mesophile | Thermophile | Psychrophile
```

We create the record type organisme_t. It will contain the record type taxonomie_t, the record type ecotype_t, the organism ID and the latin name of the organism :

```
# type organisme_t = {id_O:int; nom:string; taxonomie:taxonomie_t;
ecotype:ecotype_t};;
type organisme_t = {
  id_O : int;
  nom : string;
  taxonomie : taxonomie_t;
  ecotype : ecotype_t;
}
```

We create the record type genome_t. It will contain the genome ID, the length of the genome, the number of genes identified and the date of the last sequencing of the genome :

```
# type genome_t = {id_G:int; taille:float; genes:int; date:int};;
type genome_t = { id_G : int; taille : float; genes : int; date :
int; }
```

We defined 7 organisms. They will be in the list organisme :

```
let org_1 = {id_O = 1; nom = "Escherichia_coli"; taxonomie =
Bacterie; ecotype = Mesophile};;
let org_2 = {id_O = 2; nom = "Saccharomyces_cerevisiae"; taxonomie =
Eucaryote; ecotype = Mesophile};;
let org_3 = {id_O = 3; nom = "Arabidopsis_thaliana"; taxonomie =
Eucaryote; ecotype = Mesophile};;
```

```
let org_4 = {id_O = 15; nom = "Sulfolobus_solfataricus"; taxonomie =
Archee; ecotype = Thermophile};;
let org_5 = {id_O = 16; nom = "Thermotoga_maritima"; taxonomie =
Bacterie; ecotype = Thermophile};;
let org_6 = {id_O = 17; nom = "Pyrococcus_furiosus"; taxonomie =
Archee; ecotype = Thermophile};;
let org_7 = {id_O = 20; nom = "Caenorhabditis_elegans"; taxonomie =
Eucaryote; ecotype = Mesophile};;
```

We try some queries to check if the organisms have been recorded in the memory, here are the answers of queries :

```
val org_1 : organisme_t =
  {id_O = 1; nom = "Escherichia_coli"; taxonomie = Bacterie;
   ecotype = Mesophile}
val org_2 : organisme_t =
  {id_O = 2; nom = "Saccharomyces_cerevisiae"; taxonomie =
Eucaryote;
   ecotype = Mesophile}
val org_3 : organisme_t =
  {id_O = 3; nom = "Arabidopsis_thaliana"; taxonomie = Eucaryote;
   ecotype = Mesophile}
val org_4 : organisme_t =
  {id_O = 15; nom = "Sulfolobus_solfataricus"; taxonomie = Archee;
   ecotype = Thermophile}
val org_5 : organisme_t =
  {id_O = 16; nom = "Thermotoga_maritima"; taxonomie = Bacterie;
   ecotype = Thermophile}
val org_6 : organisme_t =
  {id_O = 17; nom = "Pyrococcus_furiosus"; taxonomie = Archee;
   ecotype = Thermophile}
val org_7 : organisme_t =
  {id_O = 20; nom = "Caenorhabditis_elegans"; taxonomie = Eucaryote;
   ecotype = Mesophile}
```

We put the 7 organisms together in a list named "organisme" : it will be the "organisme" database :

```
let organisme = [org_1;org_2;org_3;org_4;org_5;org_6;org_7];;
```

We try a query to check if the list ogranisme contains all the organisms defined before :

```
val organisme : organisme_t list =
  [{id_O = 1; nom = "Escherichia_coli"; taxonomie = Bacterie;
    ecotype = Mesophile};
   {id_O = 2; nom = "Saccharomyces_cerevisiae"; taxonomie =
Eucaryote;
    ecotype = Mesophile};
   {id_O = 3; nom = "Arabidopsis_thaliana"; taxonomie = Eucaryote;
    ecotype = Mesophile};
   {id_O = 15; nom = "Sulfolobus_solfataricus"; taxonomie = Archee;
    ecotype = Thermophile};
   {id_O = 16; nom = "Thermotoga_maritima"; taxonomie = Bacterie;
    ecotype = Thermophile};
   {id_O = 17; nom = "Pyrococcus_furiosus"; taxonomie = Archee;
    ecotype = Thermophile};
   {id_O = 20; nom = "Caenorhabditis_elegans"; taxonomie =
Eucaryote;
    ecotype = Mesophile}]
```

We try a query on the ID of the organism 1:

```
# org_1.id_O;;
- : int = 1
```

We try a query on the organism 2's Latin name :

```
# org_2.nom;;
- : string = "Saccharomyces_cerevisiae"
```

We try a query on the organism 3's taxonomy :

```
# org_3.taxonomie;;
- : taxonomie_t = Eucaryote
```

We try a query on the organism 4's ecotype :

```
# org_4.ecotype;;
- : ecotype_t = Thermophile
```

We define 4 genomes. They will be in the list named "genome" :

```
let gen_1 = {id_G = 1; taille = 4.6; genes = 4600; date = 2001};;
let gen_2 = {id_G = 15; taille = 2.9; genes = 3034; date = 2001};;
let gen_3 = {id_G = 16; taille = 2.0; genes = 1763; date = 2006};;
let gen_4 = {id_G = 17; taille = 1.9; genes = 1700; date = 2002};;
```

We try some queries to check if the genomes have been recorded in the memory, here are the answers of queries :

```
val gen_1 : genome_t = {id_G = 1; taille = 4.6; genes = 4600; date =
2001}
val gen_2 : genome_t = {id_G = 15; taille = 2.9; genes = 3034; date
= 2001}
val gen_3 : genome_t = {id_G = 16; taille = 2.; genes = 1763; date =
2006}
val gen_4 : genome_t = {id_G = 17; taille = 1.9; genes = 1700; date
= 2002}
```

We put the 4 genomes together in a list named "genome" : it will be the "genome" database :

```
let genome = [gen_1;gen_2;gen_3;gen_4];;
```

We try a query to check if the list genome contains all the genomes defined before :

```
val genome : genome_t list =
  [{id_G = 1; taille = 4.6; genes = 4600; date = 2001};
   {id_G = 15; taille = 2.9; genes = 3034; date = 2001};
   {id_G = 16; taille = 2.; genes = 1763; date = 2006};
   {id_G = 17; taille = 1.9; genes = 1700; date = 2002}]
```

We try a query on the ID of the genome 1 :

```
# gen_1.id_G;;
- : int = 1
```

We try a query on the genome 2's length :

```
# gen_2.taille;;
- : float = 2.9
```

We try a query on the genome 3's number of genes :

```
# gen_3.genes;;
- : int = 1763
```

We try a query on the genome 3's date of last sequencing :

```
# gen_4.date;;
- : int = 2002
```

We built a recursive function named search_thermophile to return all the organisms with an ecotype thermophile :

```
let rec search_thermophile liste_Org =
match liste_Org with
    |[] -> []
    |organisme_t::r  when  organisme_t.ecotype  =  Thermophile  ->
organisme_t.nom::(search_thermophile r)
    |organisme_t::r -> search_thermophile r;;
val search_thermophile : organisme_t list -> string list = <fun>
```

We tested the function :

```
# search_thermophile organisme;;
- : string list = ["Sulfolobus_solfataricus"; "Thermotoga_maritima";
"Pyrococcus_furiosus"]
```

"Sulfolobus_solfataricus" and "Pyrococcus_furiosus" are Thermophiles.

We built a recursive function named search_B4_2001 to return the numeber of genomes sequenced before 2001 :

```
let rec search_B4_2001 genome =
match genome with
    |[] -> 0
    |e::r when e.date < 2001 -> 1 + (search_B4_2001 r)
    |e::r -> (search_B4_2001 r);;
```

We tested the function :

```
val search_B4_2001 : genome_t list -> int = <fun>
```

In the example we didn't have a genome sequenced before 2001 we created one to test the function :

```
let gen_5 = {id_G = 18; taille = 5.6; genes = 5600; date = 1998};;
val gen_5 : genome_t = {id_G = 18; taille = 5.6; genes = 5600; date
= 1998}
```

We added the new genome to the « genome » list :

```
let genome = [gen_1;gen_2;gen_3;gen_4;gen_5];;
val genome : genome_t list =
  [{id_G = 1; taille = 4.6; genes = 4600; date = 2001};
   {id_G = 15; taille = 2.9; genes = 3034; date = 2001};
   {id_G = 16; taille = 2.; genes = 1763; date = 2006};
   {id_G = 17; taille = 1.9; genes = 1700; date = 2002};
   {id_G = 18; taille = 5.6; genes = 5600; date = 1998}]
```

We tested the function :

```
search_B4_2001 genome;;
- : int = 1
```

We observed that the function return only one genome

We built a recursive function named fullGenom that ask if a genome is completely sequenced by returning the genome ID :

```
let rec fullGenom i g =
match g with
    |[] -> false
    |e::r when e.id_G = i -> true
    |e::r -> fullGenom i r;;
val fullGenom : int -> genome_t list -> bool = <fun>
```

We tested the function :

```
fullGenom 1 genome;;
- : bool = true
fullGenom 15 genome;;
- : bool = true
fullGenom 3 genome;;
- : bool = false
fullGenom 20 genome;;
- : bool = false
```

We observed that we actually have the genome for the organism ID 1 and 15 but not for the organism ID 3 and 20.

We built a recursive function named Listgeno which return the list of organisms for which we got the complete genome sequenced :

```
let rec listgeno organisme genome =
match organisme with
    |[] -> []
    |e::r when (fullGenom e.id_O genome) = true -> e.nom::(listgeno
r genome)
    |e::r -> listgeno r genome;;
val listgeno : organisme_t list -> genome_t list -> string list =
<fun>
```

We tested the function :

```
listgeno organisme genome;;
-   :   string   list   =   string   list   =   ["Escherichia_coli";
"Sulfolobus_solfataricus"; "Thermotoga_maritima";
 "Pyrococcus_furiosus"]
```

We actually got the genomes of E. coli, Sulfolobus solfataricus, Thermotoga maritima and Pyrococcus furiosus.

Now we would like to know the name of which organism have 90% or more of its genome that is coding proteins, and the percentage of his genome that is coding for proteins.

We first built a function that calculate the percentage of the genome that is coding proteins for each genome we got in the genome database :

```
let calculPercentCoding genome =
(float_of_int(genome.genes)/.(genome.taille*.1000.))*.100.;;
val calculPercentCoding : genome_t -> float = <fun>
```

We tried the function :

```
calculPercentCoding gen_1;;
- : float = 100.
calculPercentCoding gen_2;;
- : float = 104.62068965517241
calculPercentCoding gen_3;;
- : float = 88.149999999999991
calculPercentCoding gen_4;;
- : float = 89.473684210526315
calculPercentCoding gen_5;;
- : float = 100.
```

Then, we built a function that will return if a genome is coding for less or more than 90 percents.

```
let moreThanNinety gene = calculPercentCoding gene >= 90.0;;
```

Then we built a function able to return all the data about a genome when a genome ID is asked :

```
let rec whichGenom i gene=
    match gene with
    |[]->failwith"Genome not found"
    |e::r when e.id_G=i ->e
    |e::r->whichGenom i r;;
val whichGenom : int -> genome_t list -> genome_t = <fun>
```

We tested the function by On test la fonction en rentrant l'id_G et le nom de la liste "genome" :

```
whichGenom 1 genome;;
- : genome_t = {id_G = 1; taille = 4.6; genes = 4600; date = 2001}
whichGenom 18 genome;;
- : genome_t = {id_G = 18; taille = 5.6; genes = 5600; date = 1998}
```

Finally we built the function that use the three functions (calculPercentCoding, moreThanNinety and whichGenom) we created previously.
This function will return a list of tuples (Latin name , % of coding) of the organisms with a genome coding for more than 90% :

```
let rec listOrgCodingMoreThanNinety genom org =
    match org with
    |[] -> []
    |e::r when (fullGenom e.id_O genom) && moreThanNinety
(whichGenom e.id_O genome) -> (e.nom, calculPercentCoding
(whichGenom e.id_O genome))::listOrgCodingMoreThanNinety genom r
    |e::r -> listOrgCodingMoreThanNinety genom r;;
val listOrgCodingMoreThanNinety :
  genome_t list -> organisme_t list -> (string * float) list = <fun>
```

Now we test that final function :

```
listOrgCodingMoreThanNinety genome organisme;;
- : (string * float) list =
[("Escherichia_coli",       100.);       ("Sulfolobus_solfataricus",
104.62068965517241)]
```

We observed that a the hypothetic genome added before does not appear in the list. So we created a corresponding hypothetic organism that we added in the list organisme :

```
let org_8 = {id_O = 18; nom = "Organisme_questiontrois"; taxonomie =
Eucaryote; ecotype = Mesophile};;
let organisme = [org_1;org_2;org_3;org_4;org_5;org_6;org_7;org_8];;
```

We executed one more time the function listOrgCodingMoreThanNinety to check if the resulting list changed :

```
listOrgCodingMoreThanNinety genome organisme;;
- : (string * float) list =
[("Escherichia_coli",       100.);       ("Sulfolobus_solfataricus",
104.62068965517241); ("Organisme_questiontrois", 100.)]
```

And we obtained the three genomes.

Now we would like to create a new function that could be used to merge 2 databases of genomes. We named it fusionBases :

```
let rec fusionBases b1 b2=
match (b1,b2) with
    |([],b2) -> b2
    |(b1,[]) -> b1
    |(e1::n1,e2::n2) when e1.id_G<e2.id_G -> e1::(fusionBases n1 b2)
    |(e1::n1,e2::n2) when e1.id_G>e2.id_G -> e2::(fusionBases b1 n2)
    |(e::n1,_::n2) -> e::(fusionBases n1 n2);;
val fusionBases : genome_t list -> genome_t list -> genome_t list =
<fun>
```

To test this new function we create two hypothetic databases genomeB1 and genomeB2 :

```
let gen_30={id_G=30; taille=6.0; genes= 3624; date=2014};;
let gen_31={id_G=31; taille=1.2; genes= 766; date=2014};;
let gen_32={id_G=32; taille=9.0; genes= 8374; date=2014};;
let gen_33={id_G=33; taille=2.3; genes= 128; date=2014};;

let genomeB1 = [gen_30; gen_31; gen_32; gen_33];;

let gen_40={id_G=40; taille=6.0; genes= 4524; date=2014};;
let gen_41={id_G=41; taille=1.2; genes= 525; date=2014};;
let gen_42={id_G=42; taille=9.0; genes= 2345; date=2014};;
let gen_43={id_G=43; taille=2.3; genes= 1234; date=2014};;

let genomeB2 = [gen_40; gen_41; gen_42; gen_43];;
```

We checked if every genomes and both lists, have been recorded in the memory :

```
val gen_30 : genome_t = {id_G = 30; taille = 6.; genes = 3624; date
= 2014}
val gen_31 : genome_t = {id_G = 31; taille = 1.2; genes = 766; date
= 2014}
val gen_32 : genome_t = {id_G = 32; taille = 9.; genes = 8374; date
= 2014}
val gen_33 : genome_t = {id_G = 33; taille = 2.3; genes = 128; date
= 2014}
```

```
val genomeB1 : genome_t list =
  [{id_G = 30; taille = 6.; genes = 3624; date = 2014};
   {id_G = 31; taille = 1.2; genes = 766; date = 2014};
   {id_G = 32; taille = 9.; genes = 8374; date = 2014};
   {id_G = 33; taille = 2.3; genes = 128; date = 2014}]

val gen_40 : genome_t = {id_G = 40; taille = 6.; genes = 4524; date
= 2014}
val gen_41 : genome_t = {id_G = 41; taille = 1.2; genes = 525; date
= 2014}
val gen_42 : genome_t = {id_G = 42; taille = 9.; genes = 2345; date
= 2014}
val gen_43 : genome_t = {id_G = 43; taille = 2.3; genes = 1234; date
= 2014}

val genomeB2 : genome_t list =
  [{id_G = 40; taille = 6.; genes = 4524; date = 2014};
   {id_G = 41; taille = 1.2; genes = 525; date = 2014};
   {id_G = 42; taille = 9.; genes = 2345; date = 2014};
   {id_G = 43; taille = 2.3; genes = 1234; date = 2014}]
```

Then we tested the function fusionBases on the two databases :

```
fusionBases genomeB1 genomeB2;;
- : genome_t list =
[{id_G = 30; taille = 6.; genes = 3624; date = 2014};
 {id_G = 31; taille = 1.2; genes = 766; date = 2014};
 {id_G = 32; taille = 9.; genes = 8374; date = 2014};
 {id_G = 33; taille = 2.3; genes = 128; date = 2014};
 {id_G = 40; taille = 6.; genes = 4524; date = 2014};
 {id_G = 41; taille = 1.2; genes = 525; date = 2014};
 {id_G = 42; taille = 9.; genes = 2345; date = 2014};
 {id_G = 43; taille = 2.3; genes = 1234; date = 2014}]
```

We created a list that contain all the lists that we create before (genome, genomeB1 and genomeB2) :

```
let listeAllBases =[genome; genomeB1; genomeB2];;
val listeAllBases : genome_t list list =
  [[{id_G = 1; taille = 4.6; genes = 4600; date = 2001};
    {id_G = 15; taille = 2.9; genes = 3034; date = 2001};
    {id_G = 16; taille = 2.; genes = 1763; date = 2006};
    {id_G = 17; taille = 1.9; genes = 1700; date = 2002};
    {id_G = 18; taille = 5.6; genes = 5600; date = 1998}];
   [{id_G = 30; taille = 6.; genes = 3624; date = 2014};
    {id_G = 31; taille = 1.2; genes = 766; date = 2014};
    {id_G = 32; taille = 9.; genes = 8374; date = 2014};
    {id_G = 33; taille = 2.3; genes = 128; date = 2014}];
   [{id_G = 40; taille = 6.; genes = 4524; date = 2014};
    {id_G = 41; taille = 1.2; genes = 525; date = 2014};
    {id_G = 42; taille = 9.; genes = 2345; date = 2014};
    {id_G = 43; taille = 2.3; genes = 1234; date = 2014}]]
```

We created a list of lists, but the lists were still not merge together.

So we create a new function (fusion_liste_base) that merged the genome lists together :

```
let rec fusion_liste_base l =
match l with
|[] -> []
|[e] -> [e]
|e::b::n -> fusion_liste_base((fusionBases e b)::(fusion_liste_base
n));;
val fusion_liste_base : genome_t list list -> genome_t list list =
<fun>
```

We tested the function on the list of lists returned previously :

```
fusion_liste_base listeAllBases;;
- : genome_t list list =
[[{id_G = 1; taille = 4.6; genes = 4600; date = 2001};
  {id_G = 15; taille = 2.9; genes = 3034; date = 2001};
  {id_G = 16; taille = 2.; genes = 1763; date = 2006};
  {id_G = 17; taille = 1.9; genes = 1700; date = 2002};
  {id_G = 18; taille = 5.6; genes = 5600; date = 1998};
  {id_G = 30; taille = 6.; genes = 3624; date = 2014};
  {id_G = 31; taille = 1.2; genes = 766; date = 2014};
  {id_G = 32; taille = 9.; genes = 8374; date = 2014};
  {id_G = 33; taille = 2.3; genes = 128; date = 2014};
  {id_G = 40; taille = 6.; genes = 4524; date = 2014};
  {id_G = 41; taille = 1.2; genes = 525; date = 2014};
  {id_G = 42; taille = 9.; genes = 2345; date = 2014};
  {id_G = 43; taille = 2.3; genes = 1234; date = 2014}]]
```

The lists have been merged together.

For the final experience we built the function idjoin that take into arguments the list organisme, the list genome and another function (fct1) that we will define further :

```
let rec idjoin organisme genome fct=
match (organisme, genome) with
    |([],_) -> []
    |(_,[]) -> []
    |(e1::n1, e2::n2) when e1.id_O>e2.id_G -> idjoin organisme n2
fct
    |(e1::n1, e2::n2) when e1.id_O<e2.id_G -> idjoin n1 genome fct
    |(e1::n1, e2::n2) -> (fct e1 e2)::idjoin n1 n2 fct;;
val idjoin :
  organisme_t list ->
  genome_t list -> (organisme_t -> genome_t -> 'a) -> 'a list =
<fun>
```

We defined the function fct1 that will look for Latin name and length of genomes respectively into the list organisme and the list genome :

```
let fct1 e1 e2 = (e1.nom,e2.taille);;
val fct1 : organisme_t -> genome_t -> string * float = <fun>
```

We tested the function idjoin on the lists genome and organisme with the function fct1 as an argument :

```
idjoin organisme genome fct1;;
- : (string * float) list =
[("Escherichia_coli", 4.6); ("Sulfolobus_solfataricus", 2.9);
 ("Thermotoga_maritima", 2.); ("Pyrococcus_furiosus", 1.9)]
```

And we succefully obtained the tuples we were looking for.