Managing and Analyzing Big-Data in Genomics

Sebastien Mondet¹, Ashish Agarwal¹², Paul Scheid¹, Aviv Madar¹, Richard Bonneau¹², Jane Carlton¹, Kristin C. Gunsalus¹

1 Center for Genomics and Systems Biology, Dept of Biology, New York University
2 Courant Institute of Mathematical Sciences, New York University

IBM Programming Languages Day 2012
Outline

1. Biology, Genetics, Big Data™, HPC
2. A DSL Approach To The Software Stack
3. OCaml, Ocsigen, ... Experience Report
Introduction
Genomics & Sequencing

Let’s over-simplify:

```ocaml
let sample = (** Some extract of formerly-living material *)
let library = (** Carefully prepared bunch of
short DNA fragments *)

let lab_tech: sample -> protocol -> library real_world_monad

let base = A | C | G | T

let read = base list
 (** one read < one previous DNA fragment *)

let sequencing: library -> reagents -> (read list) real_world_monad
```
How it looks like once on the computational side:

@SEQ 1
AATAGTAAATCCATTTGTTCAACTCAGTTTGATTTGGGTTCAAAGCAGTATCGATCA +
!''*(((***+))%%%++)(%%%%%.1***-++'''))**55CCF>>>>>>CCCCCCCC65

@SEQ 2
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAAACGATTT +
(>CCC(**!>>CCC''*((%%.1*++))%'''))**5+(5CC%+%%*-*F>>>>C65

c.f. wikipedia:FASTQ_format.
Then, a branch of computer science takes over: **Alignment** [BWA09], [Bowtie09]

```
List.map reads ~f:(fun r -> String.clever_find genome r)
```

**Assembly** [Assembly10]

```
val assemble: read list -> genome
```

**Annotation** [Annot09]

*Et cetera.*
Introduction

Numbers — Moore’s Law of Biology

Stein, Genome Biology 2010, 11:207
Introduction

Our Process

- Wet-Lab Tech.
- Prof.
- Bioinformatician
- HiSeq 2000
- Library Submission
- Website
- Remote Storage
- Transfer
- HPC Cluster + Servers
- Demultiplexing
- Statistics
- Alignment
- Prof.

Mondet, Agarwal, et al. – OCaml / Bio-seq-core
The “Layout” DSL

Why?

Need a persistence layer:

- Database → meta-data.
- File Structure → big data.

Need a more high-level representation:

- Virtual file-system: *just* a cache.
- Ensure coherency: DB tables Vs OCaml code Vs File-system.
- SQL is not portable, weakly typed, verbose.
- Quick and safe migrations.
The “Layout” DSL

The DSL Approach

Define a Domain Specific Language:

- “Our” concepts: functions, records, …
- Better typing: enumerations, non-null by default, file-system elements …
- Manageable size.
The “Layout” DSL

The DSL Approach

Then, from one single descriptive file:

- Generate the “right” SQL queries & File-system paths.
- Generate well typed OCaml code for accessing the data.
- Generate pretty graphs.
- Handle migrations peacefully.
- Provide an *introspection-like* API.
- Generate safety and consistency checking functions.
- Well-formed backups.
The “Layout” DSL

An Example — The Source

```
(volume certificate certificate_files)
(record ssl_certificate
   (expiration timestamp option)
   (file certificate))
(enumeration role user admin visitor auditor)
(record person
   (name string option)
   (login string)
   (certificates ssl_certificate array)
   (roles role array))
(* ... *)
(function aligned_data bowtie_aligner
   (genome genome)
   (phred_style phred_score_kind)
   (prng_seed int option))
```
The “Layout” DSL
An Example — The Pretty Graph

person
name: String option
login: String
certificates: ssl_certificate array
roles: role array

ssl_certificate
expiration: Timestamp option
file: certificate

phred_score_kind =
| q33
| q4

bowtie_aligner
genome: genome
phred_style: phred_score_kind
prng_seed: Int option
aligned_data

aligned_data
sam_files: sam

sam
.../sam_aligned_reads/

Mondet, Agarwal, et al. – OCaml / Bio-seq-core
The “Layout” DSL

An Example — Generated Code

module Enumeration_role = struct
  (** The type of \{i role\} items. *)
type t = ['user | 'admin | 'visitor | 'auditor] with sexp
let to_string : t -> string = function
| 'user -> "user"
| 'admin -> "admin"
| 'visitor -> "visitor"
| 'auditor -> "auditor"
let of_string_exn: string -> t = function
  (* ... *)

let of_string s =
  try Ok (of_string_exn s) with e -> Error s
The “Layout” DSL
An Example — Generated Code

```ocaml
module Record_person = struct
  type pointer = { id: int} with sexp
  type value = {
    name : string option;
    login : string;
    certificates : Record_ssl_certificate.pointer array;
    roles : Enumeration_role.t array} with sexp
  type t = {
    g_id : int;
    g_created : Timestamp.t;
    g_last_modified : Timestamp.t;
    g_value: value} with sexp
```

Mondet, Agarwal, et al. – OCaml / Bio-seq-core
The “Layout” DSL
An Example — Generated Code

```ocaml
module Function_bowtie_aligner = struct
  type pointer = { id: int} with sexp
  type evaluation = {
    genome : Record_genome.pointer;
    phred_style : Enumeration_phred_score_kind.t;
    prng_seed : int option} with sexp
  type t = {
    g_id : int;
    g_result : Record_aligned_data.pointer option;
    g_inserted : Timestamp.t;
    g_started : Timestamp.t option;
    g_completed : Timestamp.t option;
    g_status : Enumeration_process_status.t;
    g_evaluation: evaluation} with sexp
```
The “Layout” DSL
An Example — Generated Code

```ocaml
  pointer ->
  (Record_ssl_certificate.pointer,
   [> ‘Layout of Layout.error_location
    * Layout.error_cause ]) Flow.t

let add_value ~expiration ~file ~dbh =
let v = { expiration; file; } in
let work_m =
  let query =
    Sql_query.add_value_sexp ~record_name:"ssl_certificate"
    (Record_ssl_certificate.sexp_of_value v) in
  Backend.query ~dbh query
  >>= (function Some id -> return {id} |
       None -> error (‘wrong_add_value))
  in
  bind_on_error work_m (fun e ->
    error (‘Layout ((‘Record ”ssl_certificate”: error_location),
      (e : error_cause))))
```
module Bowtie_aligner = struct
  let add_evaluation ~genome ~phred_style ?prng_seed ~dbh =
      (* ... *)
  let get ~dbh pointer =
      (* ... *)
  let get_all ~dbh =
      (* ... *)
  let set_started ~dbh p =
      (* ... *)
  let set_failed ~dbh p =
      (* ... *)
  let set_succeeded ~dbh ~result p =
      (* ... *)
The "Layout" DSL

An Example — Migrations & Backups

Write

```ocaml
val migrator: S-Expression -> S-Expression
```

```bash
$ hitscore dump-to-file backup_v42
$ ./migrator backup_v42 backup_v43
$ hitscore wipe-out-database
$ hitscore init-database
$ hitscore load-file backup_v43
$ hitscore verify-layout
```
The “Layout” DSL

Our Current Layout
The “Layout” DSL

Introspection-like API

**GENCORE: THE LAYOUT NAVIGATOR**

The Layout

- Enumeration process_status
- Record log
- Enumeration role
- Record person
- Record organism
- Record sample
- Volume protocol_directory
- Record protocol
- Enumeration barcode_provider
- Record custom_barcode
- Record stock_library
- Record key_value
- Record input_library
- Record lane
- Record flowcell
- Record hiseq_run
- Record invoicing
- Volume bionalyzer_directory
- Record bionalyzer
- Volume agarose_gel_directory
- Record agarose_gel
- Record inaccessible_hiseq_raw
- Volume sample_sheet_csv
- Record sample_sheet
- Enumeration sample_sheet_kind
- Function assemble_sample_sheet
- Volume bcl_to_fasta_unaligned
- Function bcl_to_fasta
- Function transfer_hiseq_raw
- Function delete_intensities
- Volume dircmp_result
- Record hiseq_checksum
- Function dircmp_raw
- Record client_fastq_dir
- Function prepare_unaligned_delivery
- Volume generic_fastas_dir
- Record generic_fastas
- Function coerce_b2f_unaligned
- Volume fastx_quality_stats_dir
- Record fastx_quality_stats_result
- Function fastx_quality_stats
### The “Layout” DSL

**Introspection-like API**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_id:</td>
<td>3179</td>
</tr>
<tr>
<td>Identifier</td>
<td></td>
</tr>
<tr>
<td>g_created:</td>
<td>2011-12-13</td>
</tr>
<tr>
<td>Timestamp</td>
<td></td>
</tr>
<tr>
<td>g_last_modified:</td>
<td>2012-05-29</td>
</tr>
<tr>
<td>Timestamp</td>
<td></td>
</tr>
<tr>
<td>S-Exp:</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>print_name:</td>
<td>Sebastien Mondet</td>
</tr>
<tr>
<td>String option</td>
<td></td>
</tr>
<tr>
<td>given_name:</td>
<td>Sebastien</td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>middle_name:</td>
<td></td>
</tr>
<tr>
<td>String option</td>
<td></td>
</tr>
<tr>
<td>family_name:</td>
<td>Mondet</td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>email:</td>
<td><a href="mailto:sebastien.mondet@nyu.edu">sebastien.mondet@nyu.edu</a></td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>secondary_emails:</td>
<td><a href="mailto:seb@mondet.org">seb@mondet.org</a></td>
</tr>
<tr>
<td>String array</td>
<td></td>
</tr>
<tr>
<td>login:</td>
<td>cm4431</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The “Layout” DSL

Introspection-like API

Actions:

- You may add a new person
- You may modify this person

g_id: Identifier 3179
The “Layout” DSL

Introspection-like API

Function fastx_quality_stats

Actions:
- You may add a new fastx_quality_stats

<table>
<thead>
<tr>
<th>g_id: Identifier</th>
<th>5867</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_result: fastx_quality_stats_result_option</td>
<td>5861</td>
</tr>
<tr>
<td>g_inserted: Timestamp</td>
<td>2012-04-20</td>
</tr>
<tr>
<td>g_exectime: Timestamp option</td>
<td>2012-04-20</td>
</tr>
<tr>
<td>g_completed: Timestamp option</td>
<td>2012-04-20</td>
</tr>
<tr>
<td>g_status: process_status</td>
<td>Succeded</td>
</tr>
<tr>
<td>S-Exp: String</td>
<td>([input_dir ((ld 5849)) (option_Q 33) (filter_names (&quot;*.fastq *.fastq.gz&quot;))])</td>
</tr>
<tr>
<td>input_dir: generic_fastqs</td>
<td>5840</td>
</tr>
<tr>
<td>option_Q: Int</td>
<td>33</td>
</tr>
<tr>
<td>filter_names: String</td>
<td>(&quot;*.fastq *.fastq.gz&quot;)</td>
</tr>
</tbody>
</table>
The “Layout” DSL

Introspection-like API
The “Layout” DSL
It’s Everywhere
Ocaml in Genomics
Experience Report

We use Ocaml everywhere with:

- Jane St Core & Batteries
- Lwt
- Ocsigen
- Biocaml
- PG’Ocaml, XXMLM, Csv, Sqlite, The Cryptokit
OCaml and Asynchronous I/O
Lwt Is Also Everywhere

Writing application-servers & Web-services.

\[
\begin{align*}
\land \\
&\text{Preemptive threads + shared memory + human beings} = \bullet \oplus \ominus \\
\Rightarrow \\
&\text{Lwt:}
\end{align*}
\]

- Light-weight threads — Monadic non-preemptive I/O
- Don’t block — Don’t get preempted
- ocsigen.org/lwt/
OCaml and Asynchronous I/O
Lwt Is Also Everywhere

```
perform_some_complex_io x y z
>>= fun result ->
(* toy "shared mutable state" example: *)
let aux = !global_a in
global_a := !global_b;
global_b := result
return ()
```
OCaml and Asynchronous I/O
Lwt Is Also Everywhere

We actually don’t like exceptions:

- Embed a Result/Error monad → *Flow monad*
- Use polymorphic variants for the “error side”
  (extensible at will + exhaustive pattern check)

```ocaml
module type Flow = sig
  type ('ok, 'err) monad
  val bind : ('ok, 'err) monad -> ('ok -> ('oknext, 'err) monad) -> ('oknext, 'err) monad
  val return : 'ok -> ('ok, 'any) monad
  val error : 'err -> ('any, 'err) monad
end
```
OCaml and Asynchronous I/O
An Example — Error Management

```ocaml
let f i =
  match i with
  | 0 -> return ()
  | 1 -> error 'its_one
  | 2 -> error 'its_two
  | n -> error ('its_a_lot n)
val f :
  int -> (unit, [> 'its_a_lot of int | 'its_one | 'its_two ])
  Flow.t
```
OCaml and Asynchronous I/O
An Example — Error Management

```ocaml
bind_on_error (f 42)
  (function
   | 'its_one -> eprintf "One!\n"; return ()
   | 'its_a_lot n -> eprintf "A lot: %d\n"; return ()
```

Characters 51-59:
```
| 'its_one -> eprintf "One!\n"; return ()
```

Error: This pattern matches values of type `< 'its_a_lot of 'a | 'its_one ]`
but a pattern was expected which matches values of type `> 'its_a_lot of int | 'its_one | 'its_two ]`
The first variant `type` does not allow tag(s) `'its_two`
OCaml In The Wild
It’s Super-Cool

OCaml is not only well-typed:

- Industrial-Strength (core and libraries/frameworks).
- Hackability (Bypass tools, extend build-system).
- Objects and Polymorphic Variants.
- The Future (Coq).

It’s being improved on:

- The Marketing.
- The Programmer’s Toolkit.
Ocsigen
The Way To Go

A concentrate of awesomeness:

- **Well-Typed** web-programming
  ⇒ HTML5 **and** services **and** client code!
- Eliom_output.Caml.register*!(⇔ RPC-like programming)
- Choices do not get on your way
  DB design, templating, “there is more than one way” …
- Statically linked native webserver/application.
- js_of_ocaml: great but still limited by JS/DOM.
  *Something* is missing there …
Thanks
Any Questions?

Contacts:

- Ashish Agarwal
  ashish.agarwal@nyu.edu
  http://ashishagarwal.org/
- Sebastien Mondet
  sebastien.mondet@nyu.edu
  http://seb.mondet.org
References I


The Flow Monad

Why and What

Cooperative (or not) Threads:

- Be able to use \textit{Lwt}, \textit{Async}, or standard preemptive threading.
  \[ \Rightarrow \text{Functor over an I/O monad.} \]

Don’t like exceptions:

- Need a \textbf{Result Monad} (a.k.a. “error monad”)
  
  Already using a monad \( \Rightarrow \) Monad Transformer

```ocaml
module type Result_IO_monad = sig
    type ('ok, 'err) monad
    val bind : ('ok, 'err) monad -> ('ok -> ('oknext, 'err) monad) -> ('oknext, 'err) monad
    val return : 'ok -> ('ok, 'any) monad
    val error : 'err -> ('any, 'err) monad
end
```

Mondet, Agarwal, et al. – OCaml / Bio-seq-core
The “Layout” DSL
An Example — Generated Code

```ocaml
with_database configuration (fun ~dbh ->
  let layout = Classy.make dbh in
  layout#library#all
  >>= map_sequential ~f:(fun lib -> lib#preparator#get)
  >>=| List.dedup
  >>= map_sequential ~f:(fun prep_person ->
    prep_person#set_roles ('preparator :: prep_person#roles))
  >>= fun _ ->
  return ()
)```

Mondet, Agarwal, et al. – OCaml / Bio-seq-core