Biocaml: The OCaml Bioinformatics Library

Ashish Agarwal[1], Sebastien Mondet[2], Philippe Veber[4], Christophe Troestler[3]
[1]Solvuu, <agarwal1975@gmail.com>, [2]Mount Sinai, [3]Université de Mons
[4]Laboratoire de Biométrie et Biologie Evolutive, UMR 5558, CNRS, Université Lyon 1
Project Website: http://biocaml.org. *Source Code:* https://github.com/biocaml. *License:* LGPL.

Functional Programming has long served academic studies of software engineering and is now gaining traction in industrial settings. OCaml is a Functional Programming language that also has strong support for traditional imperative and object-oriented programming. Biocaml is OCaml's bioinformatics library, analogous to BioPerl, BioPython, etc, and has been used in several large genomics projects. Its current feature set can be split into 3 broad categories: i) parsing/printing of many data formats, ii) data structures on integer intervals, and iii) clients to public data repositories. Cross-cutting features include: asynchronous methods for concurrency, rigorous error handling, and comprehensive API documentation.

i) Parsers and printers for about 15 file formats are currently implemented, including: fastq, fasta, gff, sam, and bam. All can be zipped, and all support streaming. A format specification system, inspired by MIME types but more flexible, allows precise handling of all the minor variations in formats. ii) Integer intervals arise frequently in genomics. Biocaml provides sets and balanced binary trees with several variations, e.g. overlapping intervals are allowed or not, and nodes of a tree carry values or only the leaves. These serve as the basis for several other modules: memory efficient integer sets, ROC curve computations, gene model representations, and histograms. iii) Bioinformaticians rely on a multitude of public data repositories. Biocaml provides a client for several of the Entrez databases, and we hope to continue adding more.

Genomics algorithms are often IO intensive and access to remote data sources can lead to long delays. Biocaml provides an asynchronous API from the ground up; every system call is made in a non-blocking fashion. This can speed up your programs by allowing some computations to continue while other parts are waiting for a system call to return. Such concurrency handling is normally done by using an event loop or with callbacks, but Biocaml employs a concurrency monad, which allows this kind of code to be written more safely and more easily. System calls and many other computations are prone to errors. Biocaml takes error handling very seriously, again by using a monad. Sometimes, the majority of a function's implementation regards error handling.

We aim to accommodate a spectrum of programmers: simple script writers and software engineers building industrial strength applications. Thus, we do not consider it reasonable to impose monadic programming on all of our users. For system calls, a simpler but blocking API is also provided. Fortunately, the burden on Biocaml's developers is minimized due to OCaml's functors, which provides the alternate API for free. For errors, an alternate exception-ful API is provided, which, although not free, is not difficult to provide. Combined with our focus on documentation, we hope all of these features will allow Biocaml to be widely used.