# ECS240 Progress Report

Camille Welcher

Mar 4, 2016

## Parsing Grammars

Thus far, the majority of work has centered around finding a suitable EBNF grammar implementation for input and output. The appropriate library for this application will:

- Support streaming; many robust implementations, such as lex/yacc (Mason and Brown 1990), Bison ("Bison" 1988) and Boost Spirit, which is actually a Parsing Expression Grammar library (Guzman and Kaiser 2009), are meant to work in-memory, which is unsuitable when there may be hundreds of terabytes of data;
- Be very fast; the implementation should approach the speed of hand-written parsers, in order to incentivize users to actually include the validator in their pipelines;
- Be portable; this is likely an unrealistic requirement, as my research thus far indicates that these grammars tend to require semantic actions which will be language-specific. Regardless, portability would greatly enhance utility.

I experimented with Spirit a fair amount, implementing several parsers, but after initially being optimistic about its potential, it is increasingly apparent that it will be very difficult to force it in to a streaming model. Because of this, I will likely make use of Ragel ("Ragel" 2014) long-term.

However, currently I'm more interested in getting out a working prototype than finding the perfect set of libraries. With that in mind, I will make use of Biopython (Cock et al. 2009) for the remainder of the project, as it supports a wide variety of formats and is well-tested, if slow and cumbersome.

## Validation System

Significant work still needs to be completed in regards to the validation mini-language. So far, I have mostly just sketched out the interface. I will be making

gratuitous use of named pipes; the user will specify any number of named pipes to create on invocation, which can then be used later on in the pipeline. For example, a simple invocation will have the form:

```
validator --assert-subset --format fasta --in example.fasta \
--fifo output.fifo --out output.fasta | trim-reads --out output.fifo
```

This way, the validator gains access to the output of the target program without having to anything too hacky. I hope to find a less verbose way to do this, as I find this method confusing, but I've yet to think of more concise way of approaching it. `--assert-subset` is somewhat self-explanatory: this would be the most simple validation method, requiring that the output be a subset of the input.

The next week will will revolve around formally defining the validation mini-language and getting out a rudimentary prototype. After that, I will focus on expanding its compatibility to a few more formats and writing up the results.

## References


"Bison." 1988. Free Software Foundation, Inc. https://www.gnu.org/software/bison/.

Cock, P. J. A., T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, et al. 2009. "Biopython: Freely Available Python Tools for Computational Molecular Biology and Bioinformatics." *Bioinformatics* 25 (11): 1422–23. doi:10.1093/bioinformatics/btp163.

Guzman, Joel de, and Hartmut Kaiser. 2009. "Boost Spirit." http://boost-spirit.com/home/.

Mason, Tony, and Doug Brown. 1990. *Lex &Amp; Yacc*. Sebastopol, CA, USA: O'Reilly & Associates, Inc.

"Ragel." 2014. Colm Networks. http://www.colm.net/open-source/ragel/.