Bioinformatics, 33(21), 2017, 3502–3504
doi: 10.1093/bioinformatics/btx306
Advance Access Publication Date: 9 May 2017
Applications Note

OXFORD

Databases and ontologies

BioMake: a GNU make-compatible utility for declarative workflow management

Ian H. Holmes[1,2,*] and Christopher J. Mungall[3,*]

[1]Department of Bioengineering, University of California, Berkeley, CA 94720, USA, [2]Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA and [3]Environmental Genomics and Systems Biology Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

*To whom correspondence should be addressed.
Associate Editor: Janet Kelso

Received on December 12, 2016; revised on April 1, 2017; editorial decision on April 18, 2017; accepted on May 8, 2017

Abstract

Motivation: The Unix 'make' program is widely used in bioinformatics pipelines, but suffers from problems that limit its application to large analysis datasets. These include reliance on file modification times to determine whether a target is stale, lack of support for parallel execution on clusters, and restricted flexibility to extend the underlying logic program.
Results: We present BioMake, a make-like utility that is compatible with most features of GNU Make and adds support for popular cluster-based job-queue engines, MD5 signatures as an alternative to timestamps, and logic programming extensions in Prolog.
Availability and implementation: BioMake is available for MacOSX and Linux systems from https://github.com/evoldoers/biomake under the BSD3 license. The only dependency is SWI-Prolog (version 7), available from http://www.swi-prolog.org/.
Contact: ihholmes + biomake@gmail.com or cmungall + biomake@gmail.com
Supplementary information: Feature table comparing BioMake to similar tools. Supplementary data are available at *Bioinformatics* online.

1 Introduction

The familiar Unix GNU Make utility has become a favored tool for 'bioinformatics in-the-large' (Parker *et al.*, 2003). Alongside more elaborate workflow management systems, GNU Make holds its own for several reasons. Besides being ubiquitous and easy to use, with a simple syntax, it offers a powerful mix of *declarative logic* (the specification of target-dependency relationships from which Make deduces build chains) with *Unix scripting* (the lines of shell script that are executed when the build chain runs). GNU Make combines these elements with *functional programming*-inspired manipulation of file and directory names, and includes Guile — GNU's Scheme interpreter — as an extension language.

In our usage of GNU Make for data analysis, a common pattern is to analyze one or two examples manually, building up a `Makefile` recipe (or recipes), then scale the analysis up to the whole dataset. Makefiles remain, in our opinion, unrivalled for this purpose. However, GNU Make's origins were as a tool for managing build pipelines, not large-scale data analyses, and it has several flaws that impede its use in bioinformatics. For example, its use of file timestamps as a test of staleness can be fragile on networked filesystems; it lacks support for job-queueing systems; and its model of data type relies exclusively on very limited filename pattern-matching.

2 Results

We have developed a new tool, BioMake, that keeps the best features of GNU Make (including the ability to read most GNU `Makefile` syntax, with a few exceptions documented on the project's homepage) while addressing its shortcomings. Chief innovations of BioMake include:

1. **MD5 signatures as an alternative to time-stamps.** GNU Make uses file modification times to determine when files need to be rebuilt. This is fragile, especially on networked filesystems or cloud storage, where file timestamps may not be preserved or

synchronized. In projects where a big data analysis can take hours or days, a spurious rebuild can be devastating, especially if it triggers further rebuilding of downstream targets. Instead of using timestamps, BioMake can be directed to use MD5 checksums: whenever a target is built, the MD5 hashes of that file and its dependents are recorded and stored. This can be used in combination with `Makefile` recipes that sort or canonicalize data to guard against spurious rebuilds.

2. **Support for cluster-based job queues.** GNU Make can run multiple jobs in parallel, but only on one machine. It is possible to write cluster support directly into the `Makefile`, wrapping each recipe with a call to a job submission script, but this spoils GNU Make's otherwise clean separation of concerns and often prevents it from tracking dependencies properly. BioMake has built-in support for Sun Grid Engine, PBS, and SLURM job submission systems, including dependency tracking (ensuring a target is not built until all its dependents have been built). It also (like GNU Make) offers built-in parallel execution on the same machine that BioMake is being run on.
3. **Multiple wildcards per filename.** GNU Make only allows a single wildcard ('stems') in a filename, represented by the percent symbol (%) in the head of a recipe and by the automatic variable `$*` in the body. In contrast, BioMake allows multiple wildcards: any unbound variable that appears in the head of a recipe can serve as a wildcard, and can subsequently be used in the body of the recipe.
4. **Easy integration with ontologies and description logics.** GNU Make's domain-specific language extensions are based on Scheme, which is a functional language, but the underlying structure of a `Makefile` (rules such as '*to build A, you must first build B*' and '*to build B, you must first build C and D*') is a logic program. BioMake's domain-specific language is Prolog, making it easy to incorporate ontologies and description logics such as the Gene Ontology (Blake, 2015) or the Sequence Feature Ontology (Eilbeck *et al.*, 2005). For example, we can create BioMake recipes for targets such as '*the whole-genome alignment for species X and Y, where X is a mammal and Y is a vertebrate*' or '*the GFF file containing co-ordinates of every human genomic feature of type T, where T is a term descended from 'biological-region' in the Sequence Ontology*'. In a Scheme program, we would have to write, test, and debug functions that explicitly generated these lists of terms and taxa; in a Prolog database, logical conditions such as '*X is a mammal*' or '*T is descended from biological-region*' are easy to model directly, and the Prolog interpreter itself searches for all variable bindings that fit the model.

```
prolog

sp(mouse).
sp(human).
sp(zebrafish).

ordered_pair(X,Y) :- sp(X),sp(Y),X@<Y.

align_filename(F) :-
  ordered_pair(X,Y),
  format(atom(F),"align-~w-~w",[X,Y]).

endprolog

all: $(bagof F,align_filename(F))

align-$X-$Y: $X.fa $Y.fa { ordered_pair(X,Y) }
    align $X.fa $Y.fa > $@
```

Fig. 1. A hypothetical BioMake `Makefile` that runs `align` on all ordered pairs of files `mouse.fa`, `human.fa` and `zebrafish.fa`. The rule for file `align-$X-$Y` creates an alignment (using the program `align`, assumed to exist on the user's `PATH`) from any two files `$X.fa` and `$Y.fa`. However, it only applied for those `$X` and `$Y` which are flagged as being valid species, via the Prolog facts `sp(X)` which appear between `prolog` and `endprolog` directives. The top-level target `all` uses BioMake's `$(bagof...)` function, a wrapper for the Prolog predicate `bagof/3`, to find all ordered pairs of species that match the rule. This example is dissected in the repository's `README.md`

The `Makefile` in Figure 1 illustrates multi-wildcard pattern-matching (point 3, above) and Prolog extensions (point 4). This could be used to build all alignment files whose names match the pattern `align-X-Y` where `X` and `Y` are recognized species names.

3 Discussion

We can contrast BioMake with other solutions for bioinformatics workflow management. Systems such as CWL[1] or Galaxy[2] have many useful features such as web interfaces and cloud support, but they do not deduce the workflow from a graph of dependencies: rather, they require explicit specification of connections between tasks. Other GNU Make derivatives, offering features such as functional extension languages (Erlang make[3]), MD5 signatures (omake[4], makepp[5]), multiple wildcards (SnakeMake[6]) or cluster-based parallelism (e.g. Oracle Grid Engine's qmake[7]), partially overlap with BioMake, but none offers the same feature set. A comparison table is included in the Supplementary Information.

BioMake is complementary to other applications of Prolog in bioinformatics: Blipkit is a Prolog toolkit for logic programming on ontologies (Mungall, 2009). PRISM is a probabilistic dialect of Prolog used to implement graphical models for sequence annotation (Have and Mørk, 2014; Mørk and Holmes, 2012).

A natural future direction would be to develop BioMake for virtualized cloud environments, complementing its current cluster-oriented batch-processing approach to parallelism.

Funding

IHH was partially supported by NHGRI grant R01-HG004483. CJM was partially supported by Office of the Director R24-OD011883 and by the Director, Office of Science, Office of Basic Energy Sciences, of the US Department of Energy under Contract No. DE-AC02-05CH11231.

Conflict of Interest: none declared.

References

Blake,J.A. *et al.* (2015) Gene Ontology Consortium: going forward. *Nucleic Acids Res.*, **43** (Database issue), D1049–D1056.

1 `http://commonwl.org/`, accessed Dec 9, 2016.
2 `https://usegalaxy.org/`, accessed Dec 9, 2016.
3 `http://erlang.org/doc/man/make.html`, accessed Dec 9, 2016.
4 `http://omake.metaprl.org/`, accessed Dec 9, 2016.
5 `http://makepp.sourceforge.net/`, accessed Dec 9, 2016.
6 `https://snakemake.readthedocs.io/en/stable/`, accessed Dec 9, 2016.
7 `http://gridscheduler.sourceforge.net/htmlman/htmlman1/qmake.html`, accessed Dec 9, 2016.

Eilbeck,K. *et al*. (2005) The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biol*., **6**, R44.

Have,C.T., and Mørk,S. (2014) A probabilistic genome-wide gene reading frame sequence model. In: *International Work-Conference on Bioinformatics and Biomedical Engineering*, pp.350–361. Granada, Spain.

Mørk,S., and Holmes,I. (2012) Evaluating bacterial gene-finding HMM structures as probabilistic logic programs. *Bioinformatics*, **28**, 636–642.

Mungall,C.J. (2009) Experiences using logic programming in bioinformatics. In P. M. Hill and D. S. Warren, editors, *25th International Conference on Logic Programming*, volume 5649 of *Lecture Notes in Computer Science*, pages 1–21, Pasadena, CA, USA. Springer. DOI 10.1007/978-3-642-02846-5, ISBN 978-3-642-02846-5.

Parker,D.S. *et al*. (2003) Evolving from bioinformatics in-the-small to bioinformatics in-the-large. *Omics*, **7**, 37–48.