Prolog bioinformatic pipelines: a case study in gene dysregulation

Nicos Angelopoulos* and George Giamas

Department of Surgery and Cancer, Division of Cancer, Imperial College London, Hammersmith Hospital Campus, Du Cane Road, London W12 ONN, UK.

August 22, 2014

Abstract

It has been argued before that Prolog is a strong candidate for research and code development in bioinformatics and computational biology. This position has been based on both the intrinsic strengths of Prolog and recent advances in its technologies. Here we strengthen the case for the deployment and penetration of Prolog into bioinformatics, as well as describing a specific bioinformatics pipeline. Our pipeline deals with a central bioinformatics task, that of grouping and visualising a list of target genes.

1 Introduction

Prolog's traditional playground is that of knowledge representation and AI applications on crisp, logical inference and search. In addition to being a research tool in these areas, Prolog implementations have been developing to full fledged general purpose programming environments. These developments have start shaping a role for logic programming in a variety of new areas.

Bioinformatics has been been the meeting point of number of influences since its emergence as a field of study. Being on the intersection of biology, statistics and computing, it has meant that a multitude of languages, systems and paradigms has been developed and utilised for bioinformatics research. One of the strongest components in the landscape comes from the statistics community in the shape of the R (R Core Team, 2014) system and its Bioconductor (Gentleman et al., 2004) bioinformatics suite. The strength of these statistical tools is on providing a versatile platform that can incorporate a menagerie of paradigms and programming styles.

*Email: `nicos.agnelopoulos@imperial.ac.uk`

Using the knowledge representation powers of Prolog in bioinformatics has been noted before, (see for example (Angelopoulos et al., 2013; Santos Costa and Vaz, 2013)). In this paper we will discuss the suitability of Prolog for constructing bioinformatics pipelines. In particular, we will explore the traditional strength and the role of relevant recent developments in making Prolog a powerful tool for exploring important biological questions.

We describe and present results from a Prolog pipeline in the field of gene hit-lists analysis which is a central theme in bioinformatics. A number of experimental set-ups such as microarray and proteomic approaches produce a list of genes that are either up or down regulated relatively to a control condition. Experimentalists are then interested in exploring the biological signature of such lists. Two commonly undertaken analyses are the over-representation of gene ontology terms (GO, (The Gene Ontology Consortium, 2000)) and the identification of known interaction between corresponding proteins in interaction networks such as the String database (Franceschini et al., 2013). Often, in settings such as screening, there is a number of gene lists each corresponding to a part of the screen that further complicate the analysis. Our pipeline performs GO analysis, String network recovery and GO term based Sting network recovery for either a sequence or a single gene list.

The rest of the paper is structured as follows. Section 2 discusses features of, and modern developments in Prolog systems that make them suitable candidates for bioinformatics code development. Section 3 describes the development of a pipeline for the exploration of gene hit lists. Section 4 briefly gives some insights to the technical aspects of our implementation and Section 5 has the concluding remarks.

2 Logical bioinformatics

The traditional strengths of logic programming along with developments in modern implementations of the Prolog programming language provide a strong basis for implementing research led pipelines in exploratory computational biology and bioinformatics. In doing computational research it is hard to quantify the effect of the platform and programming language to the scientific conclusion. However, it is becoming increasingly clear, that the ability to reason at the highest possible level with the available data has a profound effect on the kind of questions scientists can address. It is indeed the view of experimental data as knowledge which is becoming an important realisation among the scientists in the biological sciences. Traditionally, data analysis has been viewed as an independent exercise it is increasingly becoming apparent though, that data analysis is now taking a more central role with important high-impact papers focusing on evaluating results rather than producing unique datasets.

In this context, the knowledge representation heritage and the relational foundation of logic programming are a valuable asset that is increasingly well complemented by pragmatic extensions to modern Prolog systems. Here we highlight the aspects for and against Prolog in building project code in the biological sciences. Our discussion focuses on the SWI-Prolog system (Wielemaker et al., 2012) and on lesser extend to YAP (Costa et al., 2012). SWI-Prolog is considered to be the most popular general purpose Prolog implementation. It is widely regarded as a stable, well maintained system with the most complete set of features and an active users-forum. YAP is regarded as one of the fastest Prolog implementations. The two systems are compatible in many aspects including important areas such as the interface to the *C* language and share a common repository Wielemaker and C. (2011). Two pertinent incompatibilities are the threads and web libraries of SWI which are unstable or not-implemented in YAP. On the other hand YAP has accomplished tabling infrastructure and experimental user-defined indexing extensions Santos Costa and Vaz (2013).

2.1 Positive aspects

Clear computational paradigm
Logic programming provides a single straight forward abstract machine which has a powerful dual reading. The declarative and procedural readings fit well to the task at hand. In contrast, *R* offers a multitude of competing and at times conflicting paradigms across the spectrum: functional, object oriented and procedural.

Knowledge representation and reasoning
The relational nature of Prolog along with its data-as-terms, logical variables, unification and backtracking constitute a powerful and tested knowledge representation and inference framework.

Robustness
A number of open source Prolog implementations, including SWI and YAP are now mature stable systems in which is rare to find critical bugs.

Memory management
Memory management has been a central feature of Prolog well before it was discovered by other computational paradigms. The system dynamically allocates memory positions for the program and its data, with the garbage collector automatically reclaiming chunks of the memory that are no longer needed.

Program and data duality
In Prolog there is little distinction between the program (source code) and the data. A program is simply data formed by clauses represented as terms, and which have a certain interpretation. The ability to view programs as data has fuelled important research areas such as meta-programming and program analysis. In bioinformatics, the ability to model data as programs allows to effectively represent biological knowledge. Facts can be used to map between biological entities such as mapping between genes and proteins or represent the edges of a protein-protein interactions graph.

High level of abstraction

Prolog code can be followed easily and maintained with less effort.

Calling R functions

Recently a robust library, *Real*, has been introduced that allows efficient communication from Prolog to *R*, (Angelopoulos et al., 2013). This enables the main control to remain with Prolog while at the same time the wealth of statistical functions in *R* are available to be called on Prolog data.

Indexing

Memory based data can be represented as facts in which case indexing, which has been a long standing standard Prolog technology, can be a computational asset leading to elegant and efficient code. Although traditionally indexing only considered the functor of the first argument of a predicate, recent developments show a clear interest in extending this at the knowledge base level (Santos Costa and Vaz, 2013) and at the term level (Morales and Hermenegildo, 2014).

Database interactions

The amount of available biological data has been growing extremely fast. Databases of both raw experimental results and aggregated data such as the String protein-protein interactions resource (Franceschini et al., 2013) are increasing in number and amount of data they hold. Prolog's relational nature allows for external databases to be integrated seamlessly and there exist robust packages for connecting Prolog systems to database management software. Available libraries include a multi-database interface via the *ODBC* standard (Wielemaker, 2014) and a multi-platform interface to the installation free *SQLite* database via the *proSQLite* library (Canisius et al., 2013).

Operating systems support

Practical aspects of interacting with the underlying operating system can be dealt within Prolog as modern systems provide a wide range of primitives for interacting with the host's file system. For *unix*-flavoured systems, there is further support for launching and interacting with any OS executable via the `process_create/3` primitive.

2.2 Negative aspects

Graphics and graphical user interface.

Traditionally Prolog has been a text based interactive system. Access to graphics and graphical user interfaces have been very limited. Via *Real* it is now possible to produce evolved graphical output from Prolog data. However, there is still no satisfactory way to produce graphical user interfaces for Prolog applications.

Lack of programmers.
: Although Prolog used to be taught in the computer science degrees of many universities as part of their AI course, the current trend is for fewer and fewer universities to be offering Prolog in their curriculum. In addition, there is no clear path between AI education and bioinformatics.

Language development and stability
: Prolog systems, particularly the open source kind discussed in this paper, have been small group or single person efforts. The community does not have a forum or committee to oversee and incorporate new extensions and impose standards. The core of Prolog language does benefit of a stable ISO standard (ISO, 1995) but this have not been extended to the modern and non-core features. The Prolog commons (The Prolog Commons Group, 2014) is an implementors forum that attempts to bridge gap that has yet to make significant impact. Developments in computational biology suggest that the existence of a well integrated team steering development can lead to substantial benefits to the spread of a programming language.

Critical mass
: Prolog lacks critical mass in interest, programmers, expertise on the interface with statistics and biology, and code base.

Re-usable code
: Part of Prolog's AI legacy is that although researcher did release their code, still traditionally has been little up to date, maintained, re-usable code available, no less so, due to the fragmentation in Prolog systems. One recent positive development against this trend is the package manager of SWI-Prolog, (Wielemaker et al., 2012, Appendix A.21).

Performance
: Prolog is, in general terms, considered an inefficient language. In the pipeline described here and for the type of tasks we are considering this is not the case. There will always be tasks that are performance critical and inefficient code, however, in computational biology many current challenges lie on encompassing the vast nomenclature and nuances of our current understanding of biology.

3 Gene lists

In many high throughput settings such as micro array and shotgun proteomic experiments, lists of up and down regulated genes or proteins are produced that need further down stream investigation in order to elucidate the biological significance of the experiments. Our pipeline was constructed in the context of a comprehensive screen of knock downs followed by SILAC proteomics (Zhang et al., 2014). The full pipeline clusters the knock downs on hit list similarities. Here we concentrate on the analysis of cluster-wide hit lists which is applicable to many similar settings. An example of a cluster signature is shown in Figure 1.

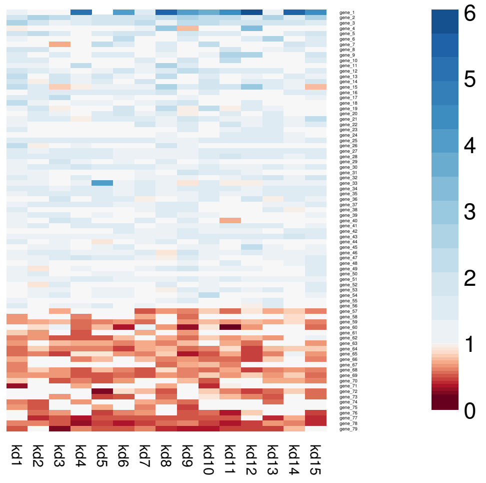

Figure 1: Downstream effects for one of the 10 clusters. Red is for down regulated and blue for up regulated genes. The numbers are log_2 fold changes from control where 1 stands for no change. On the x-axis are the knock downs in the cluster and on the y-axis are the genes significantly dysregulated

Our hit analysis combines interrogation of the gene ontology (GO, (The Gene Ontology Consortium, 2000)) and the String protein-protein interactions database (Franceschini et al., 2013). Common GO analysis includes the high level split of target genes into GO categories and the discovery of GO terms in which targets are over-represented. These are popular steps in establishing biological function and ease to use on-line tools such as David (Huang et al., 2009) allow non-experts to run the underlying programs via easy to use interfaces. We produce both types of GO output. Figure 2 shows the results of the second level terms of the biological process (BP) section of the gene ontology. In this instance there are 10 lists displayed one for each cluster the prior steps created. The percentage of genes within each cluster seen at each GO term provide a combined functional signature that is particularly informative especially when seen in comparison to the signature of other clusters.

The second type of GO analysis is the identification of GO terms that contain more hit-list genes than expected by chance. Our pipeline uses the hypergeometric test as implemented in the GOstats bioconductor library (Falcon and Gentleman, 2007). In addition to providing the usual list of GO terms that are over-represented, we mapped list genes appearing in each over represented term onto the String database of protein interactions. The database records protein-protein interactions derived from a number of diverse sources. These include links that were experimentally verified, computationally predicted or mined from the literature. The database assigns overall scores $(0 - 999)$ as well as category scores and it is possible to trace back the source of the interaction.

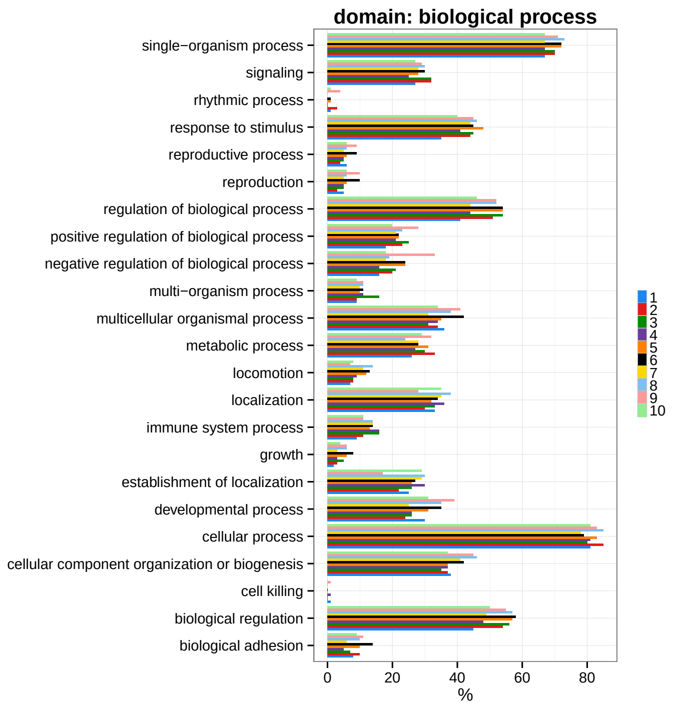

Figure 2: Gene ontology: biological process level 2. For each of the 10 clusters its percentage membership in each of the level 2 biological process terms in gene ontology hierarchy, is shown as a horizontal bar.

In our code a default value of an overall score of 500 is used. That threshold can be changed by a user specified value.

Figure 3 shows an example of a network derived by mapping the genes of a GO term. Intersecting the gene ontology term membership with the cluster gene list shows 11 dysregulated genes that are implicated in this GO term. These are mapped to expressed proteins and all their interactions are searched for in String. Red nodes are the up regulated proteins in the cluster, and green nodes depict down regulated proteins.

The main steps in constructing the GO term networks are:

(a) select significant GO terms

(b) identify genes that are implicated in each GO term

(c) map genes to proteins

(d) select from String the interactions connecting all proteins in each group

(e) generate and visualise the graph via *Graphviz*

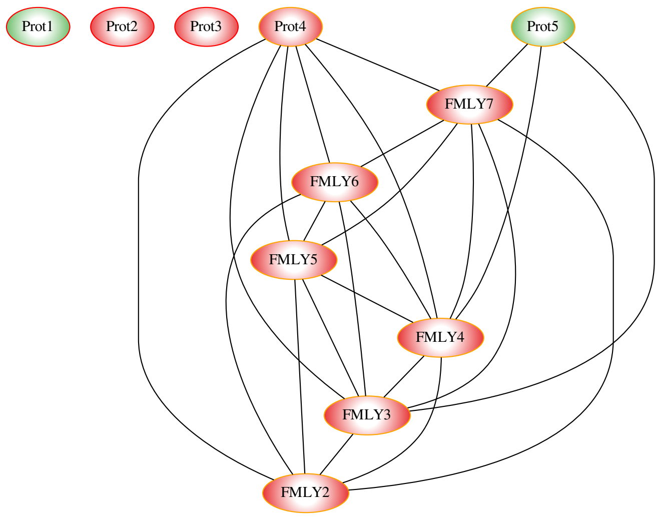

Figure 3: String network for a gene ontology biological process term.

Networks such as that of Figure 3 show the coordinated effect specific biological experiments have on known neighbourhoods of interaction. By restricting the neighbourhood to that of a GO term we focus on a well studied, curated biological process, whereas identifying String interactions between the corresponding proteins we succinctly and graphically summarise the effect of the experiment on this biological process. For instance, in the example shown, the knock downs considered, down regulated most of the FMLY family members.

4 Implementation

Here we describe some of the technical aspects of our implementation. Our approach is to construct a modular pipeline based on re-usable components. This has served us well as a paradigm previously. Indeed, we extensively use *Real* which is a component co-developed within a similar software and research paradigm. *Real* provides a robust interface to *R* which is heavily utilised in the code described here. Key to the success of such a paradigm is the discipline to identify opportunities and construct re-usable components.

Basic steps in our pipeline are separated by matrix-like data structures that are piped through the different stages. We have built a polymorphic set of predicates that can work with matrices in a variety of formats. These include basic read and write to comma separated value (CSV) files via SWI's CSV library (Wielemaker et al., 2012, Appendix A.9), in-memory fact based matrices or term lists in which each list element holds a matrix row. The in-house library defines a number of operations that operate on such matrices independently of which exact type is used as input or output. Operations include selection of rows by values or meta-calls, column separation as well as aggregation and conflation of columns to new derived ones. We hope to publish this as public software both as a stand alone library and a SWI package that can be downloaded and installed from within Prolog with a single command.

We map Entrez gene identifiers to HGNC KA et al. (2013) symbols by creating Prolog fact maps. Indexed on the first identifier, these predicated provide efficient one-directional identifier conversion. To map genes to proteins we used the `org.Hs.egENSEMBLPROT` function of the (Carlson, 2014) `org.Hs.eg.db` Bioconductor library. This package provides mappings to a multitude of entries in human biology. *Real* provides an easy way to access these and simple wrappers allow mappings to become available in Prolog.

Protein-protein interactions from *String* are represented as facts. We store both directions of each edge as to enable fast access to the neighbours of each node via first argument indexing. Visualising the String networks was done via *Graphviz* (Gansner and North, 2000). *Graphviz* provides a graph language (*dot*) and programs for producing graphics of the graphs defined in that language. As part of this and associated pipelines we have produced self-contained code that reads and writes *dot* files. In addition Prolog graph representations can be read and displayed via Prolog. Our library is not only useful in the context of String graphs, but can also be used to display graphs in the standard *ugraphs* Prolog library ((Costa et al., 2012, Section 7.26)).

5 Conclusions

We have argued that Prolog is a powerful language for building bioinformatics pipelines and that its role can be of crucial importance as biological data is increasingly needed to be viewed as knowledge both in the contexts of analysis and that of statistical inference or machine learning. Prolog's knowledge representation credentials are highly relevant in this context.

We have shown how constructing such pipelines can inspire re-usable modules that can lead to increasing use of the language in bioinformatics and computational biology. The specific pipeline discussed here, can with a reasonable amount of work be made available as a web tool using the web capabilities of SWI-Prolog (Wielemaker et al., 2008).

We also expect to make both this pipeline and the independent components described in this paper publicly available as we have previously done for *Real* (Angelopoulos et al., 2013) and *proSQLite* (Canisius et al., 2013). These include working with data in matrices, parsing and generating dot files, and extracting String sub-graphs for a list of genes or proteins.

The awareness and engagement of the Prolog community to encouraging a branch of study that is devoted to promoting logic programming in important application areas such as bioinformatics is key to further developments both to these areas as the knowledge requirement increases, as well as for the development of Prolog itself.

References

Nicos Angelopoulos, Vitor Santos Costa, Joao Azevedo, Jan Wielemaker, Rui Camacho, and Lodewyk Wessels. Integrative functional statistics in logic programming. In *Proc. of Practical Aspects of Declarative Languages*, volume 7752 of *LNCS*, pages 190–205, Rome, Italy, Jan. 2013. URL `http://stoics.org.uk/~nicos/sware/real/`.

Sander Canisius, Nicos Angelopoulos, and Lodewyk Wessels. ProSQLite: Prolog file based databases via an SQLite interface. In *Proc. of Practical Aspects of Declarative Languages*, volume 7752 of *LNCS*, pages 222–227, Rome, Italy, Jan. 2013.

Marc Carlson. *org.Hs.eg.db: Genome wide annotation for Human*, 2014. R package version 2.14.0.

Vítor Santos Costa, Ricardo Rocha, and Luís Damas. The yap prolog system. *Theory and Practice of Logic Programming*, 12:5–34, 1 2012. ISSN 1475-3081.

S. Falcon and R. Gentleman. Using GOstats to test gene lists for go term association. *Bioinformatics*, 23(2):257–8, 2007.

Andrea Franceschini, Damian Szklarczyk, Sune Frankild, Michael Kuhn, Milan Simonovic, Alexander Roth, Jianyi Lin, Pablo Minguez, Peer Bork, Christian von Mering, and Lars J. Jensen. String v9.1: protein-protein interaction networks, with increased coverage and integration. *Nucleic Acids Research*, 41(D1):D808–D815, 2013. URL `http://string-db.org/`.

Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11):1203–1233, 2000.

Robert C. Gentleman, Vincent J. Carey, Douglas M. Bates, and others. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004. URL `http://genomebiology.com/2004/5/10/R80`.

Da Wei Huang, Brad T Sherman, and Richard A Lempicki. Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources. *Nat. Protocols*, 4(1):44–57, 2009.

ISO. ISO/IEC 13211-1:1995. part 1: General core, June 1995.

Gray KA, Daugherty LC, Gordon SM, Seal RL, Wright MW, and Bruford EA. genenames.org: the HGNC resources in 2013. *Nucleic Acids Res*, 41:D545–52, Jan 2013.

Torbjorn Lager and Jan Wielemaker. Pengines: Web logic programming made easy. In *International Conference of Logic Programming*, 2014.

J.F. Morales and M. Hermenegildo. Towards pre-indexed terms. In *Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments 2014*, pages 79–92, 2014.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL `http://www.R-project.org/`.

Vítor Santos Costa and David Vaz. BigYAP: Exo-compilation meets udi. *Theory and Practice of Logic Programming*, 13(4-5):799–813, 2013.

The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nat. Genet.*, 25(1):25–9, May 2000. URL `http://www.geneontology.org`.

The Prolog Commons Group. Prolog-commons, 2014. URL `http://prolog-commons.org`.

J. Wielemaker and Vítor S. C. On the portability of prolog applications. In *Practical aspects of Declarative Languages*, pages 69–83, 2011.

Jan Wielemaker. SWI-Prolog ODBC interface, 2014. URL `http://www.swi-prolog.org/pldoc/package/odbc.html`.

Jan Wielemaker, Zhisheng Huang, and Lourens van der Meij. SWI-Prolog and the web. *TPLP*, 8(3):363–392, 2008.

Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. ISSN 1471-0684.

Hua Zhang, Justin Stebbing, Yichen Xu, Nicos Angelopoulos, and George Giamas. Reprogramming of TK-regulated proteome in breast cancer, 2014. Under review.