A logical approach to working with biological databases

Nicos Angelopoulos
Department of Surgery and Cancer, Imperial College, London, UK

Georgios Giamas
Department of Surgery and Cancer, Imperial College, London, UK

submitted 29 April 2015; accepted 5 June 2015

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, by introducing *bio_db*, a comprehensive and extensible system for working with biological data. We focus on databases that translate between biological products and product-to-product interactions, the latter of which can be visualised as graphs. This library allows easy access to high quality data in two formats: as Prolog fact files and as SQLite databases. On-demand downloading of prepacked data files in these two formats is supported in all operating system architectures as well as reconstruction from latest data files from the curated databases. The methods used to deliver the data are transparent to the user and the data are delivered in he familiar format of Prolog facts.

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 a 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 contestants in this field comes from the statistics community in the shape of the *R* (R Core Team, 2014) language 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.

Bridges between Prolog systems and *R* exist in two forms: (a) running the *R* executable session and communicating with it via the standard i/o streams, and

(b) connecting to an *R* shared library and exchange data and invoke functions via a *C* language based interface. The former approach is suitable for working with *R* code that depends on the executable's running environment. An example of a session based approach is *r_session* (Angelopoulos, 2013). An example of a shared library approach is *Real*, (Angelopoulos et al., 2013), which is suitable for communicating large volumes of data between Prolog and *R*.

Using an interface to *R* would be one way to access biological databases via packages such as *org.Hs.eg.db* (Carlson, 2014). However, this approach would increase reliance to *R* and create a further layer of complications. Here, we take a logical approach to incorporating biological knowledge. With the advances in modern Prolog systems in database integration (Canisius et al., 2013; Wielemaker, 2014) and indexing technologies (Santos Costa and Vaz, 2013; Morales and Hermenegildo, 2014) working with big data within Prolog is set to become an important application area for Prolog.

In this paper we describe the capabilities and design structure of an extensible library for working with and managing biological databases. Distinctive features of the package include: on-demand downloading of prepacked databases, ability to download and reconstruct databases from primary sources, single entry interface for accessing databases in 2 underlying serving mechanisms. Our library focuses on Homo sapiens databases and uses high-quality curated databases. Furthermore, it works on 2 Prolog systems, SWI-Prolog (Wielemaker et al., 2012) and YAP Prolog (Costa et al., 2012). Although our current implementation only supports SQLite databases due to their zero-configuration approach, it can be easily extended to other relational database systems. The intuitiveness of *bio_db* along with its relational design principles make it a natural way for handling biological databases in logic programming.

There are alternative ways to view this kind of data which depend on more evolved technologies Mungall (2009); Vassiliadis et al. (2009). The strengths of our approach in contrast are its intuitiveness, simplicity and the closeness of the produced data to the way the data are stored in the source databases.

The remainder of this paper is structured as follows. Section 2 presents the datasets readily available and the main mechanisms for using them. Section 3 describes the facilities for building the available datasets ab initio and how to incorporate new datasets. Section 4 shows some experimental results and example usage regarding the available datasets. Finally, Section 5 holds the concluding remarks.

2 A logical approach to big biological datasets

Data from biological experiments and data codifying biological knowledge have seen a sharp increase in the last decades due to the ever increasing number of high throughput techniques and the explosion in the number of researchers working on these areas. Here we will concentrate on two main categories of databases although the methodologies employed can be readily applied to any database, biological or otherwise.

The first category of databases we consider is that of mapping biological products

Pairwise maps		
Database	Abbv.	Description
HGNC	hgnc	HUGO Gene Nomenclature Committee
NCBI/entrez	entz	Nat. Center for Biot. Inf.
Uniprot	unip	Universal Protein Resource
GO	gont	Gene Ontology
Interactions database		
String	string	protein-protein interactions

Table 1. *Supported biological databases and data sources.*

and nomenclatures. A prime example in this area is the mapping genes to a unique gene names. Due to the decentralised way gene names are assigned, particularly in the early years of biological research before standardisation efforts took place, each gene is usually known by a number of different names, this is an example of an many-to-one mapping, from synonyms to unique gene name. Mapping proteins to genes is also many-to-one, but in this case because a single gene can be transcribed to a number of proteins. Many-to-many maps can be used to define membership to multiple sets. Maps are conveniently and efficiently implemented as Prolog facts of arity 2. The efficiency derives from first argument indexing (Warren, 1983; Aït-Kaci, 1991). When bi-directional translation is required, fact databases that reverse the order of the arguments are constructed.

A summary of the databases supported are shown in Table 1. Here we give a brief description of the databases included in *bio_db*. HGNC (Gray et al., 2015), is our primary gene naming data source. It is a curated and well cross referenced resource that is held at EBI. Each gene is assigned a unique incremental integer identifier and each current identifier is mapped to a unique symbol which is the short name for each gene. Example of symbols are: *LMTK3*, *EGFR* and *BRC1*. We will use `hgnc` to refer to both the database and the unique integer identifier field of the database. Symbols are shorted to `symb`. As can be seen in Figure 1 HGNC database entries play a central role in *bio_db*. The HGNC identifier connects to protein and gene resources, and Symbol connects to gene ontology terms and other naming conventions (previously-known-as and synonyms). The data populating many of the relation in Figure 1 come from the HGNC database which contains curated and submitted data from the other databases.

The National Center for Biotechnology Information (NCBI) makes available a large number of datasets (NCBI Resource Coordinators, 2013). Here we only incorporate their unique gene identifier. This is often referred to as gene id and was for many years the main way to uniquely refer to genes. Including this ensures that a number of tools and services can be used via translation of any of the other protein and gene fields to gene ids (here shortened to `entz` which is a reference to NCBI's Entrez on-line tool that uses gene identifiers as a gateway to its services).

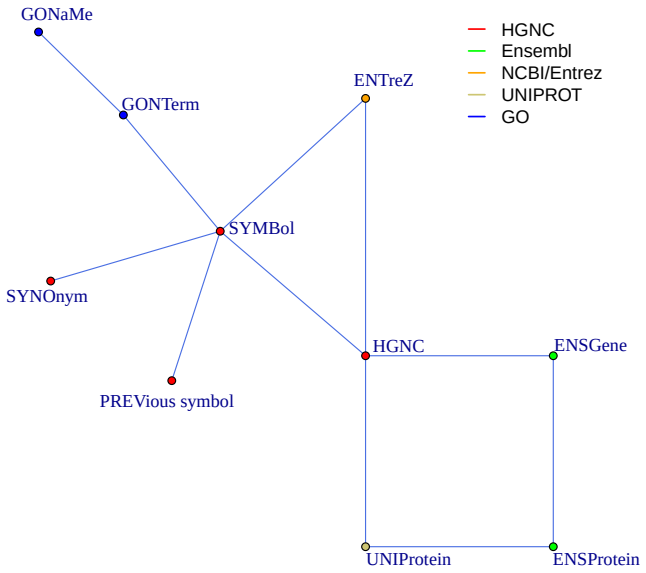

Fig. 1. Mapping predicates connect vertices of the displayed graph. The legend shows the database from which the field for each argument in the predicates is drawn from.

Uniprot (The UniProt Consortium, 2015) is a curated and well established database of proteins and related information. The relation between proteins and genes is a many to one correspondence. Each protein is transcribed from a single gene but each gene can be transcribed to more than one protein. Many biological databases record information at the protein level as this is the level at which physical interactions take place. Uniprot contains two parts, a curated resource where each protein is known to be transcribed from a specific gene and a non-curated part where not all information is complete. As our approach is gene-centric we incorporate all those proteins from both curated and non-curated parts that have an association to a gene.

Gene ontology (GO) (The Gene Ontology Consortium, 2000) provides a controlled vocabulary to describe biological knowledge. It has 3 main sections: biological process, molecular function and cellular component. The basic representation unit in GO are its GO terms. They are connected in a web of referential relations. Each term, in addition to its relative position to other terms, contains a number of genes which are involved in the process characterised by the term. Here we concentrate on this membership, which defines a many to many relation. Each term contains a number of genes and each gene can potential belong to a number of terms.

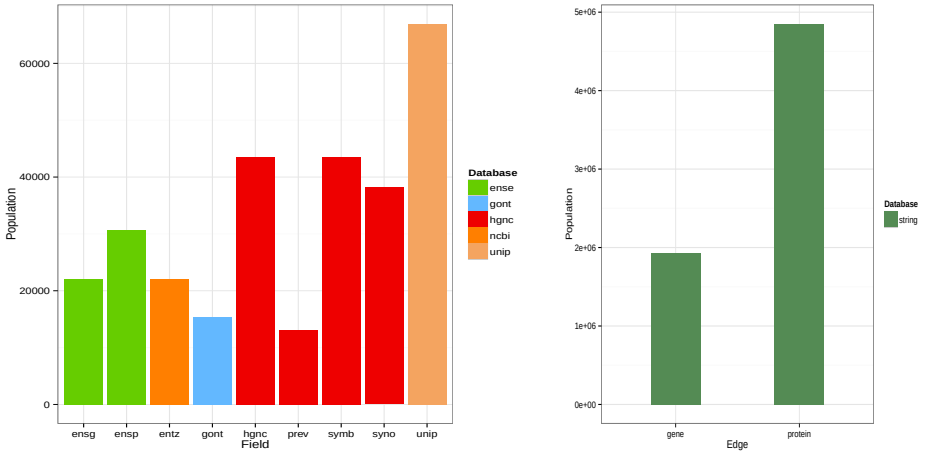

Fig. 2. Populations of the main fields in the supported databases. Each bar corresponds to a field in one of the databases. Colours correspond to databases from which the field was drawn from. In the LHS are the fields associated with maps and in the RHS are the String DB edges. The height correspond to number of items in each case and the two plots are drawn in different scales

String (Szklarczyk et al., 2015) is a comprehensive protein-protein interactions database that incorporates a large number of interactions present in one of a large number of species. Here we concentrate on the 4850628 interactions in String that pertain to human proteins (Figure 2). When mapped to symbols these form 1936162 interactions. This database collates information on each protein-protein interaction from a variety of sources such as experimental and algorithmically predicted along with publication information for papers that refer to specific links. In addition, an overall integer score in $(0, 1000)$ is provided. The closer to 1000 this score is, the more confident the curators are that this is a real physical interaction between two proteins. *Bio_db* models interactions of proteins and genes as weighed graph edges using the overall score as the weight for each edge.

3 Data management

In *bio_db* the native representation of the biological knowledge described above is as Prolog facts. The library presents those facts to the programmer as a unifying level of abstraction. Beneath this, there are two mechanisms via which the data are delivered to the predicates: (a) Prolog fact files and (b) SQLite databases.

3.1 Predicate naming

An example of a map predicate is

```
map_hgnc_hgnc_symb( Hgnc, Symb ).
```

The predicate translates between HGNC identifiers and HGNC symbols. The predicate name consists of 4 components, the first of which determines the type of data,

which in this case is a map. The second component, `hgnc` corresponds to the source database and the third component, also `hgnc`, identifies the first argument of the map to be the unique identifier field for that database (here a positive integer starting at 1 and with no gaps. The last part of the predicate name corresponds to the second argument, which here is the unique Symbol assigned to a gene by HGNC. In the current version of *bio_db*, all tokens in map predicate names are 4 characters long. The abbreviations for the database component are shown in the second column of Table 1 whereas the abbreviations for the database fields are the capitalised parts of vertice's names of Figure 1. The following interaction shows how the predicate can be used to find the symbol of a gene given its HGNC identifier.

```
?- map_hgnc_hgnc_symb( 19295, Symb ).
Symb = 'LMTK3'.
```

3.2 Data serving methods

There are two mechanisms via which the library's data predicates can be stored and served. One is as plain Prolog fact files, and the other is via SQLite databases as implemented in the proSQLite Prolog library (Canisius et al., 2013). The former requires in-memory loading for serving, thus it requires more memory and time for loading irrespective of the fact that a particular interaction with the predicate may not require the whole data set. The benefits of Prolog facts is that there are extremely fast particularly when requests for data instantiate the first argument of their call. Memory itself is in our experience not a particular limitation as computer memory is readily available in bioinformatic settings and SWI-Prolog along with most modern Prolog systems are well tuned to dealing with such data.

The time taken when loading everything to memory is a more severe limitation particularly in development settings where the data needs to be loaded a number of times in short space of time. It might thus be desirable to use SQLite during development and testing and Prolog for when big time consuming searches are required. One additional considerations is that the fact that the Prolog facts are stored in plain text files which can be helpful when debugging. Switching between the mechanisms for serving the files is done via a simple call to a predicate,

```
bio_db_interface( ?Interface ).
```

All data predicates loaded after such a call will be following the interface method dictated by `Interface`. The following example shows how the interface is switched from the default `prolog` to `prosqlite`.

```
?- debug( bio_db ).
true.

?- bio_db_interface( Iface ).
Iface = prolog.

?- map_hgnc_symb_hgnc( 'LMTK3', Hgnc ).
```

```
% Loading prolog db: .../maps/hgnc/map_hgnc_symb_hgnc.pl
Hgnc = 19295.

?- bio_db_interface( prosqlite ).
% Setting bio_db_interface prolog_flag, to: prosqlite
true.

?- map_hgnc_prev_symb( Prev, Symb ).
% Loading prosqlite db: .../map_hgnc_prev_symb.sqlite
Prev = 'A1BG-AS',
Symb = 'A1BG-AS1';
Prev = 'A1BGAS',
Symb = 'A1BG-AS1' ;
Prev = 'A1BG-AS',
Symb = 'A1BG-AS1'...
```

3.3 Downloading datasets

The library comes with placeholder code for each supported database table. On first call the relevant datafile is downloaded from the web-server and consulted on-the-fly after the place-holding code is removed. In each new interactive invocation, hot-swapping and then consulting of the relevant and data file will make the data available as facts. The facts are served transparently to the user by the two different technologies detailed above.

The downloading of non-installed datasets occurs automatically and transparently to the user. This is triggered by a call to the corresponding data predicate and the actual call is served within the same interaction as demonstrated below

```
?- debug( bio_db ).

?- map_hgnc_symb_hgnc( 'LMTK3', Hgnc ).
% prolog DB:table hgnc:map_hgnc_symb_hgnc/2 is not installed,
                          do you want to download (Y/n) ?
% Trying to get: url_file(.../map_hgnc_symb_hgnc.pl,
                          .../hgnc/map_hgnc_symb_hgnc.pl)
% Loading prolog db: .../hgnc/map_hgnc_symb_hgnc.pl
Hgnc = 19295.
```

The data files are stored in a directory organised in maps and graphs reflecting the two main type of information supported. Within these two sub directories data are organised as per database of origin. The root of this filestore organisation defaults to the data directory of the library or can be set via an environment variable or by using the `set_prolog_flag/2` predicate.

The default location for storing data files is at the level of an SWI-Prolog pack

GO term	GO name	population
GO:0003674	molecular_function	764
GO:0004674	protein serine/threonine kinase activity	340
GO:0004713	protein tyrosine kinase activity	89
GO:0005524	ATP binding	1488
GO:0005575	cellular_component	497
GO:0006468	protein phosphorylation	557
GO:0010923	negative regulation of phosphatase activity	53
GO:0016021	integral component of membrane	200
GO:0018108	peptidyl-tyrosine phosphorylation	131

Table 2. *Gene ontology terms and associated GO term names for LMTK3. Third column shows the total of genes in the GO term*

located at `pack(bio_db_repo)`. Alternatively to loading each file piecemeal, users can download the data with a single download as a pack via

```
?- pack_instal( bio_db_repo ).
```

Each dataset contains a set of house keeping information that show among other things the date the set was downloaded and built.

```
map_hgnc_hgnc_symb_info(date, date(2015, 4, 28)).
map_hgnc_hgnc_symb_info(map_type, map_type(1, 1)).
map_hgnc_hgnc_symb_info(unique_lengths, c(43592, 43592, 43592)).
map_hgnc_hgnc_symb_info(header, row('HGNC ID', 'Approved Symbol')
```

3.4 Reconstruction and new datasets

The Prolog scripts used to download and convert the data are given in the library source code. The overall work-flow normally is as follows: (a) download a remote file to a local date-stamped file, (b) read the downloaded file, (c) produce *bio_db* outputs, and (d) move or link files from downloads directory to loadables directory. These scripts can be used to reconstruct the datasets in different time points to those provided by *bio_db_repo*, thus affording more autonomy to the users.

4 Examples

Gene ontology terms are routinely used in the analysis of biological data, particularly functional analysis of target lists. For instance from a list of genes differentially expressed in an set of microarray experiments, GO term over-representation seeks to identify GO terms in which members of the differential list are present in numbers more than expected by random selection (Falcon and Gentleman, 2007).

Here we will look into the GO terms of the LMTK3 tyrosine kinase (Giamas et al., 2011). The following code shows how to produce the GO terms, their names and their populations, which are shown in Table 2.

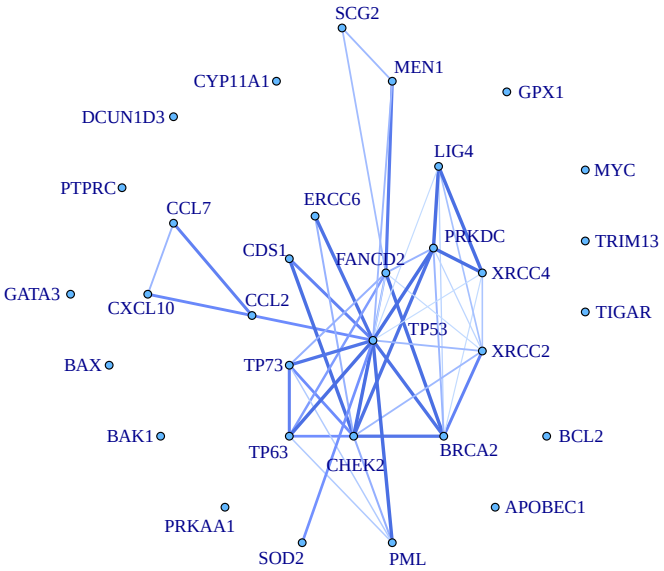

Fig. 3. Gene ontology term GO:0010332: response to gamma radiation. Edges are provided by the String database. The width and darkness of edge colour signify higher belief in the interaction being a real protein-protein interaction

```
lmtk3_go :-
   map_gont_symb_gont( 'LMTK3', Gont ),
   findall( Symb, map_gont_gont_symb(Gont,Symb), Symbs ),
   map_gont_gont_gonm( Gont, Gonm ),
   sort( Symbs, Oymbs ),
   length( Oymbs, Len ),
   fail.
lmtk3_go.

?- lmtk3_go.
```

As a second example we combine GO terms with String interactions. For a given GO term we can construct a weighted graph reflecting the interactions from the String database. This is build by first mapping an input GO term to the list of symbols it contains and then collecting all edges amongst these symbols that have a weight that exceeds that of a provided limit. The graph in Figure 3 shows such a graph for term `GO:0010332` for a minimum weight of 500.

```
go_term_graph(GoTerm,Min,Graph):-
   findall( Symb, map_gont_gont_symb(Gont,Symb), Symbs ),
   findall( Symb1-Symb2:W,  (
                          member(Symb1,Symbs),
                          member(Symb2,Symbs),
                          edge_string_hs_symb(Symb1,Symb2,W),
                          Lim < W
                             ),
                                 Graph ).

?- go_term_graph( 'GO:0010332', 500, W ).
```

4.1 Availability

The software described in this paper is available as an easy to install library for the SWI-Prolog system. Installation can be done within the system with a single call

```
?- pack_install( bio_db ).
```

This will only install the library source code but not the datasets. These will be downloaded on demand and transparently to the user upon the first call to a predicate.

All but one dataset, which have been excluded due to its size, can be also uploaded proactively with a single call,

```
?- pack_install( bio_db_repo ).
```

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 presented a library that is easily installed from within SWI-Prolog (Wielemaker et al., 2008). This library presents a convenient and intuitive way for working with biological data. All available data have been sourced from high quality and wherever possible curated databases. The emphasis of our approach is to provide easy of use, via automatically downloading datasets and using code hot-swapping, as well as flexibility by de-coupling data from code and allowing transparent ways of only downloading the necessary datasets.

There are alternative ways to view this kind of data which depend on more evolved technologies Mungall (2009); Vassiliadis et al. (2009). The strengths of our approach in contrast are its intuitiveness, simplicity and the closeness of the produced data to the way the data are stored in the source databases. Current work on the library includes extending to other databases and particularly the

Reactome database (Croft et al., 2014), as well as to other database interfaces such as ODBC. Prolog is well suited for research and code development in the areas of bioinformatics and computational biology. The code presented here, can play a strong role in promoting Prolog in these areas.

References

Hassan Aït-Kaci. The WAM: A (real) tutorial. In *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press, 1991. Also Technical report 5, DEC Paris Research Laboratory, 1990.

Nicos Angelopoulos. R_session, 2013. URL `http://stoics.org.uk/~nicos/sware/r_session/`.

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.

David Croft, Antonio Fabregat Mundo, Robin Haw, Marija Milacic, Joel Weiser, Guanming Wu, Michael Caudy, Phani Garapati, Marc Gillespie, Maulik R. Kamdar, Bijay Jassal, Steven Jupe, Lisa Matthews, Bruce May, Stanislav Palatnik, Karen Rothfels, Veronica Shamovsky, Heeyeon Song, Mark Williams, Ewan Birney, Henning Hermjakob, Lincoln Stein, and Peter D'Eustachio. The reactome pathway knowledgebase. *Nucleic Acids Research*, 42(D1):D472–D477, 2014. .

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

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`.

Georgios Giamas, Aleksandra Filipovic, Jimmy Jacob, Walter Messier, Hua Zhang, Dongyun Yang, Wu Zhang, Belul Assefa Shifa, Andrew Photiou, Cathy Tralau-Stewart, Leandro Castellano, Andrew R Green, R Charles Coombes, Ian O Ellis, Simak Ali, Heinz-Josef Lenz, and Justin Stebbing. Kinome screening for regulators of the estrogen receptor identifies lmtk3 as a new therapeutic target in breast cancer. *Nat Med*, 17:715–719, 6 2011. .

K.A. Gray, B. Yates, R.L. Seal, M.W. Wright, and E.A. Bruford. Genenames.org: the hgnc resources in 2015. *Nucleic Acids Res*, 2015.

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.

Chris Mungall. Experiences using logic programming in bioinformatics. In *Logic Programming*, pages 1–21. Springer Berlin Heidelberg, 2009.

NCBI Resource Coordinators. Database resources of the national center for biotechnology information. *Nucleic Acids Research*, 41(Database issue):D8–D20, 2013. .

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.

Damian Szklarczyk, Andrea Franceschini, Stefan Wyder, Kristoffer Forslund, Davide Heller, Jaime Huerta-Cepas, Milan Simonovic, Alexander Roth, Alberto Santos, Kalliopi P. Tsafou, Michael Kuhn, Peer Bork, Lars J. Jensen, and Christian von Mering. String v10: proteinprotein interaction networks, integrated over the tree of life. *Nucleic Acids Research*, 43(D1):D447–D452, 2015. .

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 UniProt Consortium. Uniprot: a hub for protein information. *Nucleic Acids Res.*, pages D204–D212, 2015.

Vangelis Vassiliadis, Jan Wielemaker, and Chris Mungall. Processing owl2 ontologies using thea: An application of logic programming. *OWLED*, 529, 2009.

David H. D. Warren. An abstract prolog instruction set. Technical Report 309, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Oct 1983.

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.