To appear in *Bioinformatics* (Oxford U. Press)

BioLingua: A Programmable Knowledge Environment for Biologists

JP Massar, Michael Travers, Jeff Elhai, and Jeff Shrager[1]

Abstract

BioLingua is an interactive, web-based programming environment that enables biologists to analyze biological systems by combining knowledge and data through direct end-user programming. BioLingua embeds a mature symbolic programming language in a frame-based knowledge environment, integrating genomic and pathway knowledge about a class of similar organisms. The BioLingua language provides interfaces to numerous state-of-the-art bioinformatic tools, making these available as an integrated package through the novel use of web-based programmability and an integrated Wiki-based community code and data store. The pilot instantiation of BioLingua, which has been developed in collaboration with several cyanobacteriologists, integrates knowledge about a subset of cyanobacteria with the Gene Ontology, KEGG, and BioCyc knowledge bases. We introduce the BioLingua concept, architecture, and language, and give several examples of its use in complex analyses. Extensive documentation is available online at http://nostoc.stanford.edu/Docs/index.html.

The Goal: Giving biologists hands-on capability in computational biology

In the past forty years, molecular biology has come to permeate every biological discipline. A similar revolution is in progress with regard to the massive amount of information becoming available through genomic sequencing and microarray experiments. While biologists were relatively quick to adopt the tools of molecular biology, the same is not true of the tools of bioinformatics, resulting in a growing distance between biologists and the data from which new insights must be drawn.

Although some bioinformatic tools, such as Blast, have gained popularity among biologists, combining tools and data often requires hours of painstaking database search and knowledge manipulation. In many cases biologists resort to programmers to conduct computational analyses on their behalf. Thus, the world of computational biology is accessible to biologists primarily through programmers. Whereas in the laboratory biologists can adjust their work dynamically, turning partial results over in their hands and modifying their tools to address problems and new subgoals, this flexibility is significantly reduced when programmers mediate between biologists and biological data.

BioLingua seeks to give biologists hands-on capability in computational biology by enabling them to work directly with large data sets in a familiar language, while implementation details are hidden so that programmers are needed less often. The BioLingua programming environment hides the many complexities required to implement complex knowledge-based computations by providing a wide range of databases and analytical tools, integrated with one another, and accessible through a common, simple programming language. In the BioLingua world, biologists become programmers in much the same way as molecular biologists have become enzymologists: few have isolated or characterized enzymes themselves, but they are adept at combining enzymes in novel ways. Similarly, users of BioLingua, regardless of their background, may combine the tools of computational biology, forming them in ways that are unique to the requirements of their tasks.

BioLingua embodies a number of guiding principles:

1. All required data and knowledge is accessible through a common framework. This includes genomic, metabolic, and experimental data, as well as relevant higher-level knowledge, such as the Gene Ontology (GO), and representations of metabolic pathways;

2. The system implements an efficient general purpose programming language that is augmented with biology-specific functionality, and transparent access to a rich set of basic bioinformatic tools and advanced statistical tools (e.g., R, ClustalW, MEME, the NCBI suite, etc.);

3. The environment is highly interactive, allowing experimentation and immediate reporting of results, as well as permitting long-term computations when necessary;

4. State-of-the-art systems-biology tools, such as those that assist in pathway modeling and causal interaction discovery, are smoothly integrated into the framework; and

5. The environment provides tools for collaboration among biologists regarding protocols (i.e., programs), as well as a means of announcing results and models within the community, thus enabling biologists to build upon one another's developments and discoveries.

Implementation: The Multi-Cyanobacterial BioLingua Listener (MCBL)

A BioLingua instance is a server running BioLingua loaded with knowledge for a particular community of biologists working on similar organisms. Our pilot BioLingua instance, the CIW/NASA Multi-Cyano BioLingua Listener (MCBL), tries to achieve the principles described above, and accomplishes them to varying degrees. This live web service is hosted at The Carnegie Institution of Washington, and has been publicly accessible since June of 2003. The MCBL integrates knowledge and data about a number of cyanobacteria. Several biologists are using this BioLingua instance for tasks as broad in range as bacterial genomics, causal model discovery, and dynamic analysis of biological concepts from literature. (Not all of these are working in the cyanobacterial domain.) In the remainder of this paper we briefly describe some of the principal novel aspects of the MCBL. The interested reader can review its online documentation for much more detail (http://nostoc.stanford.edu/Docs/index.html), and a demo account is available to anyone associated with a working biological laboratory.

The MCBL has several conceptual layers:

The Base Knowledge Layer represents biological component concepts, such as organism, genome, gene, protein, scaffold, etc., and their relationships. It internalizes and integrates major data and knowledge bases, such as that found in Kegg (Ogata, et al. 1999), the Gene Ontology (Gene Ontology Consortium, 2000), and BioCyc (Karp, 2001). These are locally loaded for efficient query service, and integrated with one another for convenient cross-database reference. This knowledge is captured in a knowledge-representation substrate, called a "frame system", which is described in more detail below and online.

The Computed Concepts Layer represents complex concepts built up from component concepts. Here we find concepts such as subsystem, protein complex, pathway, signal transduction pathway, etc., along with functional concepts such as enzymatic reactions. These are complex, sometimes tentative, usually context-sensitive interpretations of the primitive biological concepts rather than fixed biological facts (Shrager, 2003). Usually these are calculated by BioLingua programs which can perform arbitrary computations on the knowledge base (including the underlying sequence data) to produce complex entities such as intergenic sequences, set of orthologous proteins, and so on. Users may add their own programs to this layer, thereby creating new complex concepts. For example, a user may wish to add binding sites for a specific protein by matching the genome with a regular expression, a Markov model, or a position-specific scoring matrix.

The Programming Layer enables biologists to access and manipulate the knowledge and data in the previous layers. The BioLingua programming language, described in more detail below and online, is a derivative of Common Lisp, expanded to permit flexible integration of the various facilities inside and outside of the BioLingua environment.

The Task-Specific Interface Layer provides specific interactive graphical interfaces that drive BioLingua programs, making these available for public use. Users can easily dynamically create web-accessible interfaces to any function in the system (including their own newly programmed functions) which can be used by other users who do not have to be logged in to the BioLingua server.

Finally, the Collaboration Layer enables users to store their code and results, document their work, organize user groups, and conduct individual and collaborative activities. This layer is based upon Wiki Technology (`http://c2.com/cgi/wiki?WikiWikiWeb`).

The BioLingua system integrates many common bioinformatics tools, including ClustalW, Blast, MEME, the microarray and statistical tools from the R statistical package (http://www.r-project.org/), and others. These tools can be called directly from BioLingua, allowing them to be integrated into complex programs and dataflows. BioLingua also incorporates a "crossblast" table with the complete protein-by-protein Blast (above a certain e-value threshold) of all available genomes, and provides tools for analyzing the graph implied by this blast table. Finally, BioLingua includes a set of experimental causal modeling and discovery tools developed in one of our laboratories (derivatives of Shrager, Langley, and Pohorille, 2002; Bay, et al. 2002).

The MCBL currently integrates knowledge for *Anabaena* PCC 7120*, Anabaena variabilis* ATCC 29413*, Nostoc punctiforme, Prochlorococcus marinus* SS120, *Synechocystis* PCC 6803, and *Trichodesmium erythraeum.* Several more cyanobacterial sequences will soon be added to the database. There are currently about 260,000 concepts, i.e., frames, in the fully loaded knowledge base. The crossblast table contains about 1.4 Million entries at an e-value threshold of 1.0e-3. Concepts and tools in the layers described above are added by users and developers based on the requirements of specific tasks.

Given this rich set of tools and knowledge, it is important to understand how much work is involved in adding new knowledge or tools to the system. The maintenance process is described in detail in the online documentation (http://nostoc.stanford.edu/Docs). Here we describe it only briefly. Adding new genomes in the standard (NCBI XML) format is trivial. If they are annotated, then integration into the knowledge base is usually automatic, otherwise many tools are provided to support semi-automatic heuristic annotation of un-annotated ORFs. (We recently added a new genome (*prochlorococcus*) in about an hour, although semi-automatic annotation and updating the crossblast table take several hours.) Interfacing new computational tools is usually simple. If the tool produces XML output, parsing and re-integration of the results into the knowledge space is generally very easy. Otherwise, parsing can be somewhat of a chore, but as the underlying Lisp language is quite efficient, and provides the usual string processing tools (such as regular-expression parsing), this is no more difficult than it would be in any language, and significantly easier than in some. The most difficult process is, as one might imagine, integrating new knowledge bases into the knowledge layer, for example, integrating an entirely new pathway knowledge base. The heuristics for threading the new knowledge base with the existing knowledge backbone must be initially done by knowledge engineers (e.g., the present authors). Once the threading algorithm is in place for a particular knowledge base, re-importing new versions is usually completely automatic. The knowledge management tools provided by Lisp and the frame system are quite rich, so it is usually simple to implement the threading between the existing knowledge and a new knowledge base once the knowledge engineers have decided how it should be done.

Availability: The MCBL (http://nostoc.stanford.edu/Docs) is implemented in Franz Common Lisp (ACL 6.2, Franz Inc. Berkeley, CA) and runs on a dual Athlon Linux server (Fine Tech, San Jose, CA). It comprises approximately 50,000 lines of Common Lisp code (including internal comments) and about 15,000 lines of external documentation. The system is available under an open source public license on SourceForge (http://sourceforge.net/projects/biolingua/).

Integrated Knowledge Environment

The current MCBL combines the complete GO, the basic KEGG ontology, and a set of pathway models from BioCyc. Most permanent knowledge in BioLingua is represented in a frame system (Minsky, 1975). Frames are data structures that represent biological concepts or objects. They contain named properties (called "slots") that contain values which may include other frames or sets of frames. Some examples of frames from BioLingua include:

`#$synechocycstis6803`	The organism *synechocystis PCC* 6803
`#$s6803.slr1234`	The synechocystis gene called slr1234
`#$Go.HexokinaseActivity`	The Gene Ontology concept for the hexokinase reaction (Fig. 4)
`#$Mol.D-Hexose`	The D-Hexose molecule
`#$Go.ReactantIn`	A slot containing a set of molecules that indicates related reactions

A variety of special BioLingua forms and functions create, modify, and manage frames. Syntax extensions to the programming language permit reference to the slots of a frame to be written as: `(#^slot #$frame)`, which accesses the value of the slot #$`slot` in the frame #$frame. For example, the expression: `(#^proteins #$synechocycstis6803)` produces the proteins of this organism. A knowledge browser enables users to conveniently explore the network of frames, select desired objects and execute computations on them (Fig. 4).

Frame systems are a powerful and general way to represent knowledge. A frame represents a concept as a computational object, but unlike more standard object-oriented programming languages, frames allow slots to be defined and added dynamically during the course of a computation. BioLingua's frame system also allows the definition of slots that compute their value on demand, which provides a basic mechanism on which many different types of inference mechanisms can be built. This allows frame systems to implement richer forms of representation and inference than more restricted languages such as OWL, at cost of possibly greater computational complexity.

Knowledge bases integrated into BioLingua are threaded into one another, and are also tied to the organism models. For example, the GO and KEGG knowledge bases are threaded together through shared chemistry, and the GO is threaded into the organisms through their genes. As a result, one can implement quite complex analyses very simply, as demonstrated in the examples below. There are numerous interesting complexities involved in accomplishing the intercalation of different knowledge bases, such as the GO and KEGG. For example, there are subtle differences in the meaning of terms and relations from one knowledge base to another. These issues range from the mundane (BioCyc calls water "water", whereas KEGG calls it "H_2O"), to very complex representational issues.

An example of an especially interesting issue arises in the definition of systems of reactions and protein complexes such as photosynthesis. Different knowledge bases may choose to represent the reactions implemented by protein complexes at different, incompatible, levels of abstraction. For example, the Gene Ontology represents photosynthesis in very general terms, and its definitions are merely text strings, whereas BioCyc represents photosynthetic reaction chemistry in some detail. This problem is a symptom of a deep problem in biological functional representation (Shrager, 2003), and it may have no general solution. We have approached this problem by parsing the Gene Ontology molecular function (i.e., "activity") definitions into the molecular constituents for the left and right sides of each reaction, and adding these back to the GO frames. (Details of this are given at http://nostoc.stanford.edu/jeff/jeff/mbcs/go2.html) A new knowledge space, called "Mol" is created to contain the molecule objects that result from this parse, and then the molecule frames from BioCyc, KEGG, and the thus augmented GO are threaded into this new molecular space. As a result, BioLingua users can conveniently navigate this space, and more importantly, BioLingua programs can easily move between them in implementation of complex algorithms.

The BioLingua Language and Listener Interface

The BioLingua language is built on top of Common Lisp with a number of biological specializations. We chose Lisp because it is the most efficient language in which to write complex symbolic reasoning algorithms, and thus it is becoming increasingly important in computational biology. Lisp is also a very flexible language, with many features that permit customization to particular uses or domains. For example, Lisp has a very powerful macro capability which allows one to flexibly extend the language's syntax. (See http://www.paulgraham.com/avg.html for an explanation of why this is possible and powerful.)

Users interact with BioLingua through a web-based "Listener", which accepts either programs or commands, displays results, and keeps a running history of interactions. It is a naturally interactive language, and this sort of interaction will be familiar to users of Unix-style operating-systems and scripting languages. What is most novel is the use of a live listener on the web. Because all BioLingua users are sharing a common workspace on the same server, they can share code, results, and interactive state. This has been useful in complex collaborations such as remote user assistance and debugging. More subtle collaboration can be managed through the Wiki at the collaboration layer.

Example Programs and Results

It is remarkable how a few simple lines of BioLingua code can address biological questions that are difficult to approach outside of a general programming language, and biologists have already applied this power to make a number of interesting discoveries. For example, the ability of BioLingua to identify orthologs across genomes and to extract and manipulate sequences has permitted the backward extension of annotated genes within ortholog sets in a systematic effort to identify misannotated 5' ends (Elhai, unpublished). The integration of sequence, annotation, and convenient sequence search functions led to the identification of the genetic context surrounding repeated elements (Costa, Cousins, Elhai, and Lindblad, manuscript in preparation).

Below are three examples of BioLingua code useful in the solution of biological problems. It is important to keep in mind in studying these examples that these can be run live online right now by anyone with an MCBL BioLingua account. Moreover, these few examples barely stress the potential of BioLingua and the MCBL, which includes many other tools and knowledge-sources such as tools for microarray data analysis and causal model discovery. Many other example programs are given in the online documentation. But the power of this facility is not in these particular computations, rather it lies in the flexibility that BioLingua offers biologists who can easily implement very complex analyses of their own design.

Example 1: Finding the best-matching genes with a particular functional annotation, between multiple organisms: Because the frame system integrates both ontological knowledge and genomic information, and BioLingua provides convenient access to external tools such as ClustalW, we can very easily perform complex, knowledge-based operations, as exemplified in Figure 1. Here we have taken advantage of the fact that the genes of both Anabaena PCC 7120 and Synechocystis PCC 6803 have been assigned annotations in the Gene Ontology. In 1A we find all those genes annotated as both having kinase activity, and as being involved in glycolysis. In 1B we define a new function, `score-alignments`, which estimates the quality of alignments by calculating the fraction of perfect matches (indicated by *'s returned from ClustalW)[2]. In 1C we iterate that function over every pair of genes returned in 1A, and then, in 1D, sort the results into best-first order.

Example 2: Instantiating pathways models: Figure 2A depicts a BioLingua function that renders any given BioCyc pathway (bound to the variable: `pathway`) in terms of any given organism. For each reaction in the selected pathway, the program walks through each Gene Ontology frame related to that reaction. If there are genes from the particular organism among those genes, it collects these. This works by virtue of the integration of the BioCyc knowledge backbone with the Gene Ontology backbone and with the knowledge base for each individual organism.

Example 3: Using alignment to check annotations: The final example (Figure 3) is coded in an extension of BioLingua – called "BioLingua-Lite" – which provides a simpler syntax, and automates many procedures for the user. This extension was written by one of us (JE) and his students using the Lisp macro facility.

Consider the problem of trying to identify the gene encoding the enzyme cyanophycin synthetase in the cyanobacterium *Trichodesmium erythraeum*. In 3A we obtain the set of genes from all the loaded organisms which are annotated as "cyanophycin synthetase". There is no *Trichodesmium erythraeum* protein in this list, which is surely an annotation error, as this organism is known to possess cyanophycin. We could use one of the proteins on the list to Blast *Trichodesmium*, but which protein? Are they all really cyanophycin synthetases?

In 3B we use the experimentally determined partial amino acid sequence of a cyanophycin synthetase (Ziegler, et al 1988) to directly search for a true cyanophycin synthetase sequence in *Anabaena variabilis.* In 3C the orthologs of this gene are gathered, producing a set that includes p-Te?4231, a protein of *Trichodesmium*. However, this protein is annotated as "D-alanine-D-alanine ligase". In order to assess whether this protein is as similar to the proven cyanophycin synthetase as other proteins annotated cyanophycin synthetase, the set of orthologs is combined with the previously gathered set of putative cyanophycin synthetase genes. In 3D the complete set is aligned (only a typical portion of the alignment is shown to save space). It is evident that there are three families of protein sequences. The *Trichodesmium* protein, p-Te?4231, falls into the family containing the proven cyanophycin synthetase and shows 79% identity over the full extent of the protein (not shown). We conclude that the annotation of the Trichodesmium p-Te?4231 is incorrect: this protein is indeed a cyanophycin synthase.

Related Work

A number of projects aspire to the same goals as BioLingua, and implement some (or envision all) of the same underlying mechanisms. For purposes of comparison we roughly divide these mechanisms into database and tool integration, and end-user programming, although they are usually combined in complex ways in any particular project.

Database-integration systems such as Kleisli (Davidson, et al. 2001), DiscoveryLink (Haas et al 2001), and myGrid (Stevens, et al. 2003) implement integrated queries across a variety of data sources that may be distributed across networks, with local caching to improve speed. (See Karasavvas, et al., 2004 for a recent review.) Unlike these, BioLingua is an end-user programming language, not just a query language. BioLingua users can write programs that transparently pass the results of one computation on to the next, add to the knowledge base, and integrate partial results with computational tools such as Blast and R, enabling a much richer range of activity than is afforded by queries alone. Although it is possible to query knowledge bases across the web, BioLingua was conceived to run on a local knowledge base so as to permit operations with frequent data lookups to run orders of magnitude more quickly than they would over dispersed data. Therefore BioLingua provides a locally integrated knowledge base around a number of related organisms. Finally, BioLingua goes beyond most existing knowledge integration systems, which are usually based around XML or OWL (such as myGrid) in that our frame system offers more functionality. For example, one can define computed slots in the BioLingua frame system, enabling arbitrary inferences, including default reasoning, which can't be easily done in the Description Logic framework used by OWL.

While a number of biologists are actively using the MCBL in its present form[3], one of our current efforts seeks to further increase accessibility by including a visual programming language as part of the environment. There are a number of programmable bio-genomic data analysis tools that use a visual wiring/dataflow paradigm, such as PipeLine Pilot (www.scitegic.com), Gene Spring (www.silicongenetics.com), and QueryConnect (Hudson, et al., 2004). This model originated with SketchPad (Sutherland 1963) and has been commercialized by systems like ProGraph (Cox & Pietryzkowsky, 1989) and Labview (www.ni.com). Such languages are relatively easy for novices learn, but can lack flexibility and the power to manage complexity. So long as one's algorithms are naturally represented in a pure dataflow model, these work well, but when one needs to break from this model, for example, where different cycles of iteration interact with one another, the restrictions of this computing paradigm can become cumbersome. Because BioLingua is based upon a mature programming language, it has no such limitations, but it may be difficult for biologists to learn. We hope to address this problem by implementing a graphic language that translates BioLingua syntax into visual affordances based upon Behave! (Travers, 1994).

Many bioinformatic engineers currently rely upon the Bio* languages (BioPython, BioJava, etc.). Paradoxically, while the Bio* languages are good for the things programmers need to do for biologists (integration, reformatting, and interfacing), these are not the activities that biologists want to spend their time doing. Biologists want all of those tasks done for them to begin with, and to just get on with their analyses. This is why BioLingua provides an integrated environment. Once such integration has been done, these Bio* languages lose much of their utility. Darwin (Gonnet, et al. 2000, http://cbrg.inf.ethz.ch/) offers more richness in bioinformatic computations than the Bio* languages, but contrasts with BioLingua in a number of ways. Darwin is a new interpreted language, whereas BioLingua is based upon a mature compiled language, Lisp, which can be easily used interpretively as well. Therefore, BioLingua offers both more efficiency and more expressive power than Darwin, for example in terms of macros and syntax extensions. Although not strictly a string-processing language, Darwin is focused largely on sequence analysis, whereas BioLingua was designed from the outset as a knowledge processing system. Also, Darwin runs locally and must be downloaded to a user's machine, whereas BioLingua is available to anyone who can access the World Wide Web.

Finally, BioLingua's integrated, web-based environment offers much greater immediate access to the relevant knowledge than any of these languages. What biologists need is neither a *string processing* language, nor a general purpose language that the user has to manually connect to their knowledge sources. What they need instead is a *knowledge processing language* embedded in an integrated knowledge base already loaded with all the relevant knowledge and data.

Conclusion

Biologists are swimming in data and knowledge, but making efficient use of it can be a struggle. Even if the appropriate data and knowledge exist in a database accessible via a defined API, it is often a significant engineering effort to implement a new computation; data must be retrieved and reformatted and external tools must be interfaced, each in its own unique way. Before any science can be done, a programmer will often spend a huge amount of time pulling together the data and tools, finally putting together an intricate one-time-only program to do an intricate one-time-only calculation. Even then, with two people having to interact, one a programmer and one a biologist, there is great potential for error or confusion, for a problem to be overlooked, and for misunderstanding of the specification or misinterpretation of the results.

The goal of BioLingua is no less than that of returning biological computation into the hands of biologists themselves -- to enable biologists to directly manipulate biological knowledge and data in an interactive computational environment, much as they can now directly manipulate biological molecules in the laboratory. This is important for computational biology because biology is a living science. Biologists are constantly devising new questions that could fruitfully be addressed by computations over data and knowledge which are not easily (or at all) available via web/database interfaces. In reducing the need for expert programmers to help biologists do their work, BioLingua and projects like it begin to put computational biology directly into the hands of biologists.

Acknowledgments

This work was supported by a Biological Science Research Program grant from the NASA program in Astrobiology and Fundamental Biology. The advice of Pat Langley, and Andrew Pohorille was instrumental in the development of BioLingua, and in our thinking. We thank Franz Inc. and Xanalys for supplying copies of their excellent Lisp engines, Edi Weitz for CL-PPCRE, and Dan Barlow for CLiki. Stephen Bay, Julian Brookes, Monica Jain, Ashvin Kumar, Dan Siroker, and Sumudu Watugala did some of the coding in early implementations, and James Mastros helped in the creation of BioLingua-Lite. We especially thank Bob Haxo for his support of the MCBL server. Others who offered useful advice include Devaki Bhaya, Arthur Grossman, and Serdar Uckun. The paper was greatly improved by the comments of several anonymous reviewers.

Notes

1. Author's addresses: (JPM: massar@alum.mit.edu); MDL, Inc. (MT: mt@alum.mit.edu); Center for the Study of Biological Complexity, Virginia Commonwealth University (JE); and Carnegie Inst. of Washington Dept. of Plant Biology (JS: jshrager@stanford.edu). Corresponding author: Jeff Shrager: Inst. for the Study of Learning and Expertise, Palo Alto, CA; jshrager@stanford.edu.

2. Counting pairwise identities in Clustal-generated alignments is not a good way in general to rank genes by similarity. We use it here merely to exemplify some of the BioLingua facilities.

3. One of the authors (Jeff Elhai) is a biologist. Moreover, he has trained several students (graduate, undergraduate, and high school) in BioLingua. Indeed, most of the 20 or-so active users of the MCBL are biologists. Of course, these biologists are acting, in effect, as programmers. We claim that BioLingua can be effectively used by *biologist*-programmers, thereby making *non-biologist* programmers less crucial. The fact that a number of biologists are making effective use of the system provides at least an existence proof to back up this claim.

References

Bay, S., Shrager, J., Pohorille, A., and Langley, P. (2002) Revising regulatory networks: From expression data to linear causal models. *J. Biomed. Informatics*, 35, 289-297.

Cox, P.T., Giles, F.R. and Pietrzykowski, T. (1989) Prograph: a step towards liberating programming from textual conditioning. *IEEE Workshop on Visual languages*, 150-6.

Davidson, S.B., Crabtree, J., Brunk, B.P., Schug, J., Tannen, V., Overton, G.C., and Stoeckert, C.J. Jr. (2001) K2/Kleisli and GUS: Experiments in integrated access to genomic data sources. *IBM Systems Journal*, 40(2).

Gene Ontology Consortium (2000) Gene Ontology: tool for the unification of biology. *Nature Genet.,* **25:** 25-29.

Gonnet, G.H., Hallett, M.T., Korostensky, C., and Bernardin, L. (2000) Darwin v. 2.0: an interpreted computer language for the biosciences. *Bioinformatics*, 16(2):101-3.

Haas, L. M., Schwarz, P. M., Kodali, P. , Kotlar, E. , Rice, J. E. , and Swope, W. C. (2001) DiscoveryLink: A system for integrated access to life sciences data sources. *IBM Systems Journal*, 40(2).

Hudson, T., Stapleton, A., and Brown, J. (2004). Codifying bioinformatics processes without programming. *BIOSILICO*, 2(4).

Karp, P.D. (2001) Pathway Databases: A Case Study in Computational Symbolic Theories. *Science*, 293, 2040-2044.

Karasavvas, K.A., Baldock, R., and Burger, A. (2004) Bioinformatics integration and agent technology. *Journal of Biomedical Informatics*, 37(3):205-219.

Minsky, M. (1975) A Framework for Representing Knowledge. In: PH Winston (ed.), *The Psychology of Computer Vision*. McGraw-Hill, New York.

Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., and Kanehisa, M. (1999) KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res*. 27, 29-34.

Shrager, J. (2003) The fiction of function. *Bioinformatics*, 19(15), 1934-1936.

Shrager, J., Langley, P., and Pohorille, A. (2002) Guiding revision of regulatory models with expression data. *Proc. of the Pacific Symposium on BioComputing*. World Scientific Press.

Stevens, R., Robinson, A., and Goble, C.A. (2003) myGrid: Personalised Bioinformatics on the Information Grid. *Bioinformatics* 19(Suppl. 1):302-304.

Sutherland, I. E. (1963) Sketchpad -- A man-machine graphical communication system. *Proc. Spring Joint Computer Conference (AFIPS)*, 328-346.

Travers, M. (1994) Recursive Interfaces for Reactive Objects. *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*. Boston.

Ziegler, K., Diener, A., Herpin, C., Richter, R., Deutzmann, R. and Lockau, W. (1998) Molecular characterization of cyanophycin synthetase, the enzyme catalyzing the biosynthesis of the cyanobacterial reserve material multi-L-arginyl-poly-L-aspartate (cyanophycin). *Eur. J Biochem*., 254, 154-159.

Figure 1:

A
```
>> (setf gk-genes
        (intersection (#^go.related-genes #$go.glycolysis)
                      (#^go.related-genes #$go.kinaseactivity)))

:: (#$S6803.sll1196 #$S6803.sll1275 #$S6803.slr0394 #$S6803.sll0593
    #$S6803.sll0587 #$S6803.sll0745 #$A7120.alr2973 #$A7120.all4131
    #$A7120.all7335)
```

B
```
>> (defun score-alignment (genes)
     (let ((c (#^consensus (align genes))))
       (* 100.0 (/ (count #\* c) (length c)))))
```

C
```
>> (setf gk-scores
        (loop for pair in (cross-subsets gk-genes)
              collect (list pair (score-alignment pair))))
:: (((#$S6803.sll1196 #$S6803.sll1275) 29.248554)
   ((#$S6803.sll1196 #$S6803.slr0394) 40.41096)
   ((#$S6803.sll1196 #$S6803.sll0593) 42.201836)
   ((#$S6803.sll1196 #$S6803.sll0587) 33.789627)
   ((#$S6803.sll1196 #$S6803.sll0745) 53.900707)
   ((#$S6803.sll1196 #$A7120.alr2973) 39.262062)
   ((#$S6803.sll1196 #$A7120.all4131) 38.71795)
   ((#$S6803.sll1196 #$A7120.all7335) 64.72832)
[...36 lines of output in total...]
```

D
```
>> (sort gk-scores '> :key 'second)

:: (((#$S6803.slr0394 #$A7120.all4131) 70.01718)
    ((#$S6803.sll1196 #$A7120.all7335) 64.72832)
    ((#$S6803.sll0593 #$A7120.alr2973) 57.945736)
[...36 lines of output in total...]
```

Figure 1. Finding the best aligned kinase genes in glycolysis (as annotated at this time) between Synechocystis PCC 6803 and Anabaena PCC 7120. Commands are preceded by '>>', responses appear after '::'. Each may include several lines. Long outputs are abbreviated. Underlines represent links to the frame browser (as Fig. 4). (**A**) The built-in `intersection` function is used to find genes (in the currently loaded organisms: S6803 and A7120) annotated both as having kinase activity, and as being involved in glycolysis. The result is saved in the variable `gk-genes`. (**B**) We define `score-alignment`, which returns the percentage of perfect matches (*'s) in the alignment of any two genes. `Align` is a built-in function that calls ClustalW. (**C**) Align every pair of genes among those the gk-genes, returning the pair and their score. These are saved in `gk-scores`. (`Cross-subsets` is a built-in function that returns the non-redundant 2-way subsets of a set.) (**D**) The built-in `sort` function is used to sort the alignments in `gk-scores` into best-first order.

Figure 2:

A

```
>> (defun instantiate-pathway (pathway)
     (loop for reaction in (ensure-list (#^reaction-list pathway))
           as enzymes = (loop for ez-reaction in
                              (ensure-list (#^enzymatic-reaction reaction))
                            collect (#^enzyme ez-reaction))
           append (loop for enzyme in enzymes
                       collect (#^gene enzyme))))

>> (instantiate-pathway #$Phoslipsyn-Pwy)

:: (#$S6803.slr1369 #$S6803.sll1522 #$S6803.slr0054)
```

B

```
>> (loop for pathway in (#^allsubclasses #$pathways)
         collect (list pathway
                       (instantiate-pathway pathway)))
::
...
(#$Trna-Charging-Pwy
  (#$S6803.slr0220 #$S6803.slr1560 #$S6803.slr0357 #$S6803.slr1550
   #$S6803.slr0649 #$S6803.sll0454 #$S6803.sll1553 #$S6803.sll1425
   #$S6803.slr1703 #$S6803.sll0078 #$S6803.slr1884 #$S6803.ssr1720
   #$S6803.slr1031 #$S6803.slr1720 #$S6803.sll0495 #$S6803.sll0362
   #$S6803.slr0557 #$S6803.sll1074 #$S6803.sll1362 #$S6803.sll0179
   #$S6803.slr0958 #$S6803.sll0502))
...
(#$Aro-Pwy
  (#$S6803.slr2130 #$S6803.slr1559 #$S6803.sll1669 #$S6803.slr0444
   #$S6803.sll1747))
...
[Returns 145 instantiated pathways]
```

Figure 2. Interactive BioLingua session instantiating pathway models in cyanobacteria. Notation is as in Fig. 1. (**A**) The function instantiate-pathway collects the genes associated with each enzymatic reaction (defined by BioCyc) in a given pathway. Results are the three genes known to be associated (in this particular load of the knowledge base) with *Synechocystis* PCC 6803. The ensure-list function is needed to canonicalize the argument because BioCyc allows this slot to have either a list or singleton value. (**B**) Here instantiate-pathway, just defined, is called repeatedly with each known pathway (defined by BioCyc). This results in 228 instantiated pathways. (Only two selected results are shown.)

Figure 3:

A

```
>> (ASSIGN cph-annotated (GENES-DESCRIBED-BY "cyanophycin synth"))

:: (#$A29413.Av?2455 #$A29413.Av?2424 #$A29413.Av?3038
    #$A7120.alr0573 #$A7120.alr1779 #$A7120.all3879
    #$Npun.NpR1819 #$Npun.NpR2361 #$Npun.NpR5823 #$S6803.slr2002)
```

B

```
>> (LOOP FOR protein IN (PROTEINS-OF Avar)
         AS sequence = (SEQUENCE-OF protein)
         WHEN (SUCCESSFUL (SEARCH "RILKIQTL" sequence))
         COLLECT protein)

:: (#$a29413.p-Av?3038)
```

C

```
>> (ASSIGN cph-orthologs (ALL-ORTHOLOGS-OF p-Av?3038))

:: (#$Tery.p-Te?4231 #$A29413.p-Av?3038 #$A7120.p-All3879 #$Npun.p-NpR5823
    #$S6803.p-Slr2002)

>> (DESCRIPTION-OF p-Te?4231)

:: "COG1181: D-alanine-D-alanine ligase and related ATP-grasp enzymes
   [Trichodesmium erythraeum"

>> (ADD-TO-SET cph-orthologs (PROTEINS-NAMED cph-annotated))

:: (#$Tery.p-Te?4231 #$A29413.p-Av?3038 #$A7120.p-All3879 #$Npun.p-NpR5823
    #$S6803.p-Slr2002 #$A29413.p-Av?2455 #$A29413.p-Av?2424
    #$A7120.p-Alr0573 #$A7120.p-Alr1779 #$Npun.p-NpR1819 #$Npun.p-NpR2361)
```

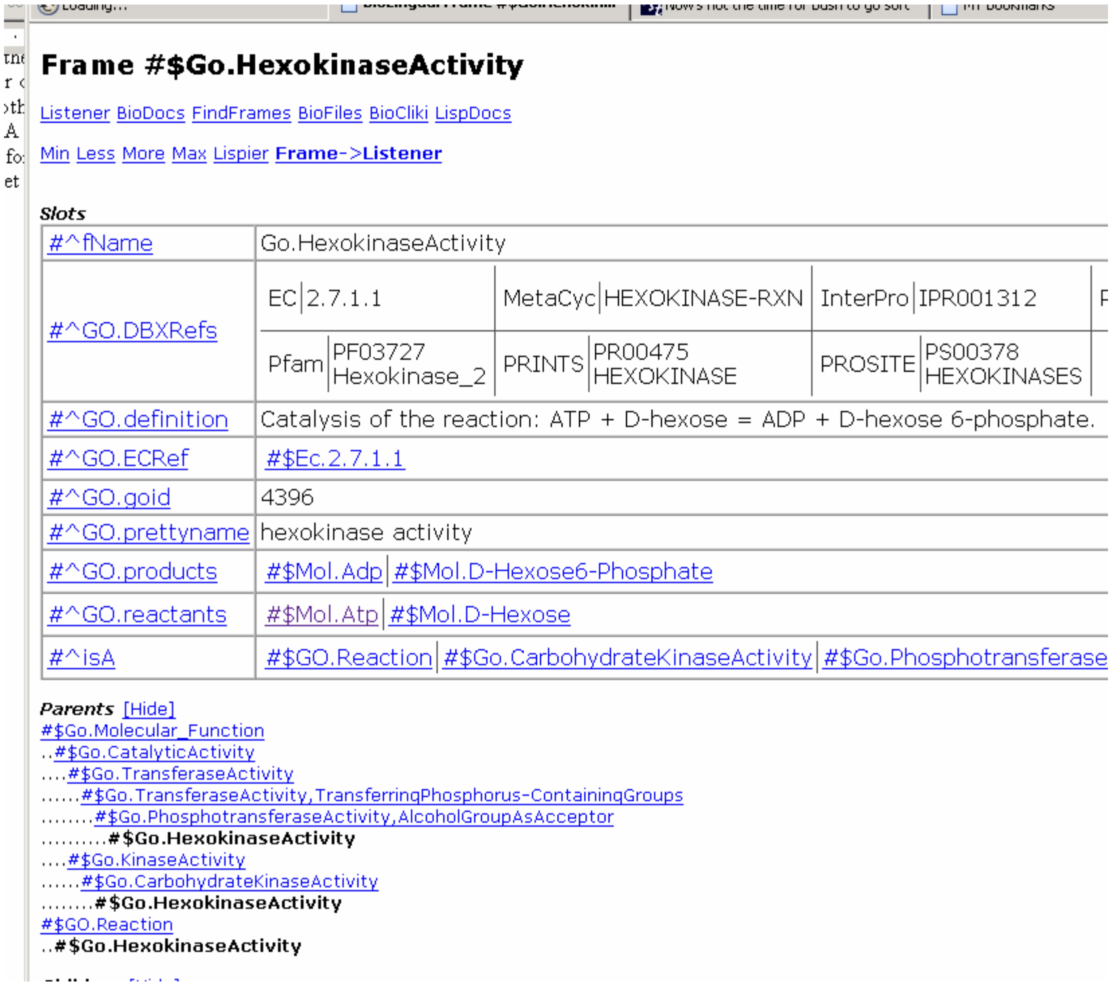
D

```
>> (ALIGNMENT-OF cph-orthologs)

Output abbreviated:
#$A29413.p-Av?2455   102 AKLIGFPVII KPINLSQGKL VTKIHNKTEY YQVAKKILRA NAG------L
 #$A7120.p-Alr1779   102 AKLIGFPVII KPINLSQGKL VTKVHNKTEY YQVAKKIFRA NAG------L
  #$Npun.p-NpR2361   104 AKELGFPVIV KPINLSQGIF VTKVHNKQEY YQIAKKILQK VSG------F
#$A29413.p-Av?3038   251 EYVGGYPIVI KPLDGNHGRG ITIDIRSWEE AEAAYEAARQ VSR-----SI
 #$A7120.p-All3879   251 EYVGGYPIVI KPLDGNHGRG ITIDIRSWEE AEAAYEAARQ VSR-----SI
  #$Npun.p-NpR5823   251 EYVGGYPIVV KPLDGNHGRG ITIDIRTWEE AEAAYEAARQ VSR-----SI
  #$Tery.p-Te?4231   251 EYVGGYPIVI KPLDGNHGRG ITIDVKNWQE AEEAYDLARK ASKTK---TV
 #$S6803.p-Slr2002   251 NDVGGYPVVI KPLDGNHGRG ITINVRHWEE AIAAYDLAAE ESKSR---SI
#$A29413.p-Av?2424   247 AKDIGYPVAV KPVVGHKGIG VTADIKDVDE LEVAYDRALE AIPENEPARI
 #$A7120.p-Alr0573   247 AKDIGYPVAV KPVVGHKGIG VTADVKDVDE LEVAYDRALE AIPENEPARI
  #$Npun.p-NpR1819   247 AREIGYPVAV KPVVGHKGIG VTADVQDSKE LESAYNRALA AIPEDQVTRI
         consensus   251     *:*: : **:   :*   :*   .  .     * .                .
```

Figure 3: Application of BioLingua-Lite to check and find an unannotated protein. Notation is as in Figs. 1 and 2. See text for details. (**A**) Defining a set, `cph-annotated`, consisting of cyanobacterial genes with annotation of "cyanophycin synthetase". (**B**) Determination of protein corresponding to experimentally proven cyanophycin synthetase from *Anabaena variabilis.* (**C**) Defining a set, `cph-orthologs`, consisting of orthologs of the proven protein. After examining the annotation of one of the genes, this set and `cph-annotated` are then combined. (**D**) Portion of alignment of candidate cyanophycin synthetases from different cyanobacteria. The sequence of the experimentally proven cyanophycin synthetase (p-Av?3038) and its orthologs are highlighted.

Figure 4:



Frame #$Go.HexokinaseActivity

Listener BioDocs FindFrames BioFiles BioCliki LispDocs

Min Less More Max Lispier **Frame->Listener**

Slots

#^fName	Go.HexokinaseActivity
#^GO.DBXRefs	EC 2.7.1.1 / MetaCyc HEXOKINASE-RXN / InterPro IPR001312 / Pfam PF03727 Hexokinase_2 / PRINTS PR00475 HEXOKINASE / PROSITE PS00378 HEXOKINASES
#^GO.definition	Catalysis of the reaction: ATP + D-hexose = ADP + D-hexose 6-phosphate.
#^GO.ECRef	#$Ec.2.7.1.1
#^GO.goid	4396
#^GO.prettyname	hexokinase activity
#^GO.products	#$Mol.Adp \| #$Mol.D-Hexose6-Phosphate
#^GO.reactants	#$Mol.Atp \| #$Mol.D-Hexose
#^isA	#$GO.Reaction \| #$Go.CarbohydrateKinaseActivity | #$Go.Phosphotransferase

Parents [Hide]

```
#$Go.Molecular_Function
..#$Go.CatalyticActivity
....#$Go.TransferaseActivity
......#$Go.TransferaseActivity,TransferringPhosphorus-ContainingGroups
........#$Go.PhosphotransferaseActivity,AlcoholGroupAsAcceptor
..........#$Go.HexokinaseActivity
....#$Go.KinaseActivity
......#$Go.CarbohydrateKinaseActivity
........#$Go.HexokinaseActivity
#$GO.Reaction
..#$Go.HexokinaseActivity
```

LEGEND

Figure 4. A page from the BioLingua Frame Browser, depicting some of the slots and values in the Gene Ontology frame representing hexokinase activity.