To Appear in The Journal of the History and Philosophy of Science (2007)

The evolution of BioBike: Community adaptation of a biocomputing platform

Jeff Shrager

Carnegie Inst. of Washington, Dept. of Plant Biology
and Stanford University, Symbolic Systems Program (consulting)
jshrager@stanford.edu

Abstract

Programming languages are, at the same time, instruments and communicative artifacts that evolve rapidly through use. In this paper I describe an online computing platform called BioBike. BioBike is a trading zone where biologists and programmers collaborate in the development of an extended vocabulary and functionality for computational genomics. In the course of this work they develop interactional expertise with one anothers' domains. The extended BioBike vocabulary operates on two planes: as a working programming language, and as a pidgin in the conversation between the biologists and engineers. The flexibility that permits this community to dynamically extend BioBike's working vocabulary – to form new pidgins – makes BioBike unique among computational tools, which usually are not themselves adapted through the collaborations that they facilitate. Thus BioBike is itself a crucial feature – which it is tempting to refer to as a participant – in the developing interaction.

Keywords: Trading Zones, Collaboration, Interactional Expertise, Genomics, Computer Programming Languages, Lisp.

1. Having been becoming a molecular biologist[1]

Computers as technical tools pervade modern science. Uniquely flexible devices, they can simulate any imaginable instrument, and programming has become a standard skill in many sciences such as physics and chemistry.[2] Biologists realized the importance of computers only after The Web had popularized the "vending machine" model of computation, where everything is expected to be available at the push of a button. As a result, the current generation of working biologists is not trained in programming and must instead rely upon programmers who are usually not trained in biology. Indeed, whole new fields, including "bioinformatics" and "computational biology", have grown up to fill this niche.[3]

About 1996, armed only with training in computer science and cognitive science, I decided that I wanted to be a molecular biologist and work on cyanobacteria, a class of environmentally important marine photosynthetic bacteria.[4] Fortuitously, at about that time a friend asked me to join his computational drug discovery start-up, which offered me both an excuse to learn organic chemistry and biochemistry, and a possible path to the funding I would need to be able to volunteer in a lab. While commuting to that job by CalTrain I studied chemistry and biology from books, and in 1999 I took the Cell,

---

[1] This work was conducted at the Carnegie Institution of Washington, Department of Plant Biology, and supported in part by grants from the NASA Ames research Center and NSF. Harry Collins and Rob Evans gave me extensive guidance on this paper and on the concepts of trading zone and interactional expertise. Comments from Robert Crease and Evan Selinger helped to improve the paper. I am especially indebted to Mike Gorman for making me think more deeply about all of my work. Mike's sense of humor has also infected me—for the better, I think.

[2] Galison, 1997, ch. 8, provides a detailed analysis of some of the many complexities of computers as instruments and as simulations.

[3] Bioinformatics represents more of the database side of biological programming, whereas computational biology represents more of the scientific and mathematical side. One can even get vocational degrees in bioinformatics. Here I will use the term "biocomputing", to avoid the subtle differences between these.

[4] BSE/MSE 1980/81 in Computer Science, MS/PhD 1982/85 in cognitive psychology. In my PhD work I studied human learning and simulated it with computer models. I subsequently spent 9 years conducting research along those lines, and then did a post doc in cognitive neuroscience.

Molecular Biology, and Biochemistry Graduate Record Examinations. Soon after, in 2000, I volunteered in the laboratory of Dr. Arthur Grossman at the Carnegie Institution of Washington, Department of Plant Biology.

When I joined the lab I began to keep a "cognitive diary" of insights about my learning about "real" laboratory biology.[5] Although it runs only a few months, many kinds of learning are described, categorizable grossly as: (a) concrete procedures including both physical skills and ways of attending and perceiving, (b) explicit knowledge about concrete procedures, such as the reasons things work or not in various situations, (c) abstract or concrete facts and relations about organisms or biological systems, (d) extra-domain knowledge and skills, such as what procedures are preferred in which labs, which journals are best to read or submit to, how to speak deferentially to faculty v. post docs v. grad students, how not to invade people's bench and freezer space, how to keep notebooks, and (e) ways of speaking, reading, writing, and interacting. Here is an example entry that crosses a number of these types of expertise (all emphasis is as the source text):

> 20000709: [...] although I can run gels now easily, I am not yet sure what in all, or even in most cases I'm supposed to get as a result. So I asked -LZ-, and she showed me what to expect. HOWEVER, in showing me how to do this, she asked me which vector I was using, and I told her that I was using the T-Gem kit. "But which vector?" she persisted. Is there more than one?! I had only the vaguest idea of what she was talking about. So she showed me in the manual that there are two vectors, one of which is cut by EcoRI and one of which isn't. Which one had I used? Now, I had studied the manual for this kit VERY VERY CAREFULLY before beginning into this process. But it was suddenly clear to me that I hadn't understood a word that it was saying! There are two vectors?! Um, well, whatever one you handed me. As it turns out, I was lucky and had used the vector that is, in fact, cut by EcoRI, and the gel actually worked -- or, more precisely, the restriction worked -- or, even more precisely, one of the four copies of the restriction worked (I've learned to do everything in four copies because three of them won't work. I'm hoping against hope that after a few years I'll be good enough at this that I won't have to do this all the time.) So this time it happened to work out, and I learned a small but important fact about the use of this kit, which is that you have to keep track of which vector you're using.

[5] Shrager, 2000. This formed the basis of Shrager, 2004, and Sahdra and Thagard, 2003.

> But [...] nearly at the moment at which -LZ- explained to me about the two vectors, something much larger clicked into place for me. I don't know quite how this happened, but somehow I had all the pieces of the puzzle (well, this local puzzle anyhow) in hand and identified, but hadn't put them into the frame. When -LZ- showed me the picture in the manual of the two vectors, with their various restriction sites, that was the frame for the whole procedure, and all the pieces fell right into it, and I very suddenly -- literally in a matter of a few seconds -- "saw" what I had been doing for the past day: I could see why we were cutting the vector and amplifying the gene, and ligating them together and why I had to use EcoRI. And then I understood, all in that same perceptual unit, how to figure out what to expect from the gel. Maybe this was just the first time I had actually had time to think, as opposed to feverishly cooking and being lost, but it doesn't feel like that. I think that I've been trying to think all the way along, but there just wasn't enough material to think with, or there were crucial pieces missing, or the frame was missing, or something.

Much of this could be described as learning how to "talk biologist"—what Collins and Evans term "interactional expertise" [6] A more literal example of language learning is found in the following extract:

> 20000724: [Note added 20000830: Everyone says EcoRI: "Eco-R-one," (not "Eco-R-Eye") but there seems to be disagreement about some others. For example, -DB- calls BglI "Bagel-one", which sounds dumb to me. And -LZ- calls SmaI "Smaaa-one", which also sounds dumb. Generally, unless the thing is very common and very pronounceable (like EcoRI), I prefer to spell out the letters, as "B-G-L-one" or "S-M-A-one", and I've heard people do this. Generally, saying things wrong is among the most embarrassing beginner mistakes one can make. It marks you as not knowing what you are talking about, unless you are a foreigner, even though these are all made-up words, so who cares! ...]

The point to take away from these examples is that as after nearly three years of, on average, a few hours/day of study of biochemistry and biology, I got to the lab armed with a great deal of book knowledge yet still had little idea how to do or talk molecular biology. This I learned "on the job", although no doubt my studious preparation was important to my facility in picking it up at all.

[6] Collins and Evans, 2002.

2. Programmers in the lab

In the late 1990s the need for heavy duty computing broadsided academic biology.[7] As the only skilled programmer in our lab, I found myself being asked to do more and more computational work for my lab-mates. At first I resisted this, having joined the lab to work on cyanobacteria not to program (esp. if I was volunteering!), but found myself with more and more of my own computing to do. As my laboratory skills were not nearly as well developed as those of my co-workers (who had working in labs for as long as I had been programming), I was more useful to everyone—myself included—at a keyboard rather than at a microscope. Saddened, I found myself, after a year at the bench, back at a keyboard doing biocomputing instead of biology.

I did my work in the programming language with which I was familiar, Lisp, and developed a specialized set of Lisp-based tools which I called BioLisp, and in about 2001 founded BioLisp.org to make these tools public for others to use.[8] This was also around the time of the Dot-Com bust (mid 2000) and biology was perceived as "The Next Big Thing". Many software engineers found me through BioLisp.org and asked how they could become involved in biocomputing. Rather than try to train each one individually, I created a set of online tutorials to explain molecular biology to my fellow computer scientists.[9] I thought that it would be a boost to both computer science and biology if more computer scientists followed my path—some might even come to our lab and save me from having to do all the programming so I could return to the bench.

---

[7] Usually this arrived in one of three forms: 1. the need to statistically analyze the results of high though-put experiments, for example from DNA microarrays; 2. the need to use large genomic databases to study the function of individual genes or systems of genes; or 3. the need to create such databases for organisms of interest to particular labs where they did not already exist. All of these involve significant complex computation, and usually from-scratch programming.

[8] This online collection was superseded in 2005 by BioBike, described later in this paper.

[9] This series, entitled "Just Enough Molecular Biology for Computer Scientists" was online from about 2001 through 2005. Like BioLisp.org, these lessons were supersede in 2006 by BioBike.

My route to biocomputing—computer scientist to chemical informaticist (the start-up interlude) to wet lab biologist and now back to bioinformaticist—is unusual in the modern computing world. In the early days of computing, before there was a specialization called "computer scientist" or "software engineer", most programmers came from the fields in which they were programming—degreed physicists, for example, programmed the software for computational physics. But it was soon discovered that the skill set of programming is generally applicable, so most modern programmers have specifically studied programming without any domain specific expertise.[10] This seems a curious situation: Engineers developing what is often a critical part of a complex application without themselves having a deep understanding of the domain – curious, but not unusual; if one steps back and looks at collaboration in general it is often, indeed usually, the case that not all the participants in the collaboration share equal knowledge of the problem domain. I now want to turn my attention to a specific interaction of this sort—where programmers who have no specific expertise in biology interact with biologists to develop computational instruments intended to be used by biologists.

3. Biocomputational trading zones

At about the same time that I produced my online lessons for programmers wanting to migrate into biocomputing, I hatched a complementary scheme to teach the biologists to program. I saw that it was soon going to be necessary for biologists to program, just as it has been necessary for physicists, chemists, and engineers. (Moreover, I reasoned that if I could teach the biologists how to program they could do it themselves, leaving me to laboratory work.) Sometime in 2001 JP Massar, a software engineer, happened upon my online lessons and volunteered extensive editorial notes.[11] I shared with him the idea of teaching biologists to program, and a particular my vision of a web-based programming

[10] Purdue launched the world's first Computer Science degree program in 1962.
[11] I recently asked JP, who holds a BS degree in Mathematics from MIT, to describe his biological background. He says of himself: "My 7th grade science project was on DNA and RNA (took 2nd in the regional fair!) I did not take any biology in college. So basically what I'd picked up from popular science books, Scientific American, and various magazine/newspaper articles."

environment that they could use to do their work. JP and I prototyped this platform, which we called BioLingua (subsequently renamed BioBike), which came online in 2002. Shortly after BioLingua went public we began collaborating with Jeff Elhai, a biologist specializing in the same subfield as mine (cyanobacteriology). Elhai had some programming experience, and also saw the need for biologists to learn to program. He became a strong advocate for BioLingua, but wanted it to be simpler for Biologists. Elhai developed, within BioLingua, a new "biologist friendly" programming language, called "BioLite". Over the next few years, and with the assistance of several other engineers, the platform eventually encompassed all of these developments, becoming what was finally called BioBike, which is still under active development.[12]

Someday biologists may be using BioBike or something like it to carry out their own biocomputations, but it is the more immediate co-evolution of the skills and knowledge of JP and Jeff Elhai, and others who have been participating in the BioBike project that I want to understand through the present analysis. Importantly, it is not only change in *people's* skills and knowledge that are at issue here; at the same time that I was starting to work with my lab-mates on our mutual biocomputing problems and developing BioLisp, JP, Jeff Elhai, and I were learning how to collaborate with one another; just as Lisp begat BioLisp, BioLisp begat BioLingua begat BioLite begat BioBike, the participants in this history—both human and computational—adapt their skills and work practices into new patterns which persist and which ultimately, through the development of tools like BioBike, may spread into ways of doing science in a broader scientific community.

This collaboration between scientists and engineers produces what Galison calls a "Trading Zone", wherein colleagues with differing skills come together to solve technical problems.[13] Some trading zones are marked by "inter-languages"—pidgins, possibly

---

[12] Massar, et al., 2005.

[13] In analyzing the intersection of multiple scientific and engineering communities involved in the development of particle accelerators, radar, and the atomic bomb, Galison, 1997, uses the metaphor of a trading zone to describe how scientists and engineers from different disciplines can collaborate across apparently incommensurable paradigms. Through the use of jargons, then pidgins, and finally full-scale creoles it is

eventually becoming creoles. Galison focused not so much on language per se, but rather on the instruments developed by physicists[14]; for him the physical instruments and their usage represented the interlingua that support trade among collaborating theoretical physicists, experimental physicists, engineers, and others engaged in collaboration.[15] Just as Galison considered physical instruments as components of inter-languages, and as computers are now fundamental scientific instruments, I consider programs, programming languages, and programming platforms as representing components of the inter-languages that facilitate trade between collaborating biologists, and especially between and software engineers who program the computers to serve biology in this capacity. [16]

4. Programming: from the general purpose to the domain specific

Computer programs are conceptually complex artifacts: at the same time practical devices producing tangible results that go into papers, and sentences in a formal language (the programming language) that can communicate meaning. Like natural languages, programming languages change in use, but they change very rapidly; programmers create new nouns, verbs, and syntax as needed to suit a given application. This is another similarity with the inter-languages used trading zones.

---

possible for communication to take place locally even when the participants may disagree about global meanings.

[14] The idea of trading zones is generalised and schematised by Collins, Evans and Gorman (this volume), leaving Galison's inter-language trading zones as just one type whereas many of those following Galison have treated them as coextensive with the term as a whole.

[15] I am specifically avoiding many interesting questions about the "reality" of computed results. Interesting and important as these issues may be, they are tangential to the present paper.

[16] Crease, 2006, describes a similar setting in protein-folding, which is done on supercomputers, and which serves as "an interface where the culture of experimental protein biology meets that of physicists and computer engineers." He describes a class where biologists learn some programming and engineers learn some biology. BioBike differs from a formal class in that the learning is happening in situ of the computation and, as mentioned, the BioBike platform itself can be thought of as a participant in the collaboration. (This is not to say that it works in the same way as the human participants.)

These properties of programs and programming languages—that they are at the same time instruments and communicative artifacts, and they evolve rapidly through usage in a community of practice—have been taken advantage of in numerous settings to facilitate collaboration between scientist and programmers in settings called "computing platforms".[17] BioBike is one such computing platform, serving a community of biologists and software engineers.[18]

This remainder of this paper focuses on the evolution of the usage of BioBike, and the way in which it evolved through collaboration between biologists and software engineers. Before getting to the meat of the analysis, let us visit some of the objects that will be observed in the BioBike biocomputational trading zone. First, BioBike operates in the same "client-server" paradigm that will be familiar to anyone who has used The Web: Your web browser is a "client", and the service that you are working with, for example Amazon.com, is a "server". In the same way, BioBike is accessed by biologists through a standard web browser. However, BioBike differs in two important ways from web servers such as Amazon.com. First, when you are using Amazon.com you are using pre-built functionality selected from menus or activated by clicking on buttons or links; sometimes you may enter content such as search terms, your credit card number, or a book review. In contrast, users of BioBike are at all times commanding a fully-functional programming system by writing sentences in a programming language called BioLisp or BioLite. These are versions of the general purpose Lisp programming language that have been specialized for biology-specific application.

The second way in which BioBike differs from Amazon.com (and most other web services) is that its functionality is extended by its user community. Although the content

[17] The programs may also be taken as what Galison, 1997, and others have called "boundary objects" – physical entities through which participants in an interaction coordinate their activities even though their separate projects may be diverse.

[18] Other similar communities have existed, especially those based on Lambda MOO (Curtis and Nicols, 1992). A good genetic term for this sort of setting would be a "collaboratory", but this term is commonly used to mean many things, most of which are not of this sort.

of Amazon.com can be extended by its user community—if you, for example, contribute a book review—Amazon.com's functionality cannot be extended by its user community; users cannot, for example, cause Amazon.com's search algorithm to work differently. But in BioBike one *can* make changes and extensions of this sort. Indeed, this is the central difference between a programming platform and any other sort of computing system: The users can change (or extend) the algorithmics of the platform. The way that this works is subtle and important and requires understanding the nature of programs and programming languages.

Programming languages come in two varieties: General Purpose, and Special Purpose (also called Domain Specific). A special purpose programming language is nearly always just a specialization of a general purpose programming language, that is, with additional lexicon and syntax for specific domains. The interesting question is how a general purpose programming language comes to be specialized, and that is what we will examine in some detail below. The general answer is that software engineers add new domain-specific nouns (called "objects"), verbs (called "functions" or "procedures"), and sometimes new syntax (called "syntax") to the programming language. For example, to specialize Lisp to BioLisp I added functions that operate on genomic objects—for example, functions that search gene sequences (those string composed of A, C, G, and T that one sees in the background of every news report on biology these days) for patterns. This functionality has no general utility; it is useful only to molecular biologists, but it is absolutely critical to their work. By adding functions of this sort (and, of course, many others) to the general purpose Lisp programming language, I developed the domain-specific BioLisp programming language, and this is also the way in which Jeff Elhai created BioLite—a further specialization of Lisp and BioLisp. [19]

But how do the software engineers know what functionality to add? There is, of course, no single nor simple answer to this question. In my case, I was both programmer and

[19] Certain peculiar properties of Lisp, particularly its self-referential macro facility, not available in any other fully-functional programming language have made Lisp the language of choice in many communities for the development of new programming languages.

biologist; I understood what was needed. But, as described above, this is rare for present-day biology. More commonly, biologists and programmers will be involved in a collaboration where the biologists set the goals and the programmers execute on them. In this case the programmers generally do not have to understand the biology at all—they are just manipulating strings or numbers at the behest of the biologists. This is the most common current case.[20]

Language evolution in the BioBike case happens somewhat differently. The programmable client-server architecture of BioBike enables both the programmers and biologists to evolve the programming language. This is an important difference—not only are programmers extending the language, but biologist users are as well, usually with some assistance from programmers. In order for this to be effectively organized, the biologists have to learn something about programming, and the programmers have to learn something about biology – and, of course, they need to learn enough to collaborate with one another. In the BioBike community the various participants must at least learn to the level of having "interactional expertise" with regard to one another.[21]

We have seen interactional expertise at least once before; it was among the things that I learned "on the job" when I began in Arthur Grossman's laboratory (recall my diary entry on the pronunciation of EcoRI, etc.) I learned my interactional expertise by "living with"

---

[20] This reflects the discussion of the meaning of the `direct' application of an expertise found in the paper on referred expertise (this volume).

[21] Collins and Evans, 2002, defined this term in the case where a sociologist gains a certain level of conversational expertise in the domain of scientists they are studying. However, the scientists usually don't learn to talk sociology. [But, for a case where they did, see the interview with Gary Sanders in the paper on science management in this volume. The same was true of a number of other physicists in Collins's gravitational wave project, Peter Saulson being just one clear additional example (ed.)] Contrast the present case where the gaining of expertise is through this sort of collegial collaboration, so each learns the others' language to some extent. Collins says of this: "The wheels of interdisciplinarity are lubricated by interactional expertise. E.g., in my gravity wave project there are theorists who are experts on sources, experts on interferometry, experts on data analysis, and so forth. These cannot do each others work but they understand what the other is doing through (so we would claim) interactional expertise in the other areas." (personal communication).

biologists elbow to elbow in the lab. But there are no shared elbows (nor, so far as I know, other body parts) in the client-server world of BioBike; indeed, the participants are nearly always in different time-zones! The way that interactional expertise evolves in the BioBike domain is through the detailed conversations that take place in and around the BioBike trading zone, including numerous email and telephone calls, and occasionally simultaneous online sessions where several biologists and programmers are seeing what others are typing into the system and how it responds. We shall see examples of these interactions in the next section. The thing to keep in mind is that at the same time as the biologists are learning to "speak programming", and the programmers are learning to "speak biologist", the BioBike platform (which represents the pidgin inter-language in the BioBike trading zone) is being evolved.

5. The co-evolution of programming platform and expertise

In BioBike both the programming language itself and the expertises of the participants evolve in synchrony with one another. Having participated in the BioBike community both as a programmer and as a biologist since about 2002 I was able to collect extensive email communications between some of the biologists and programmers involved. My analysis draws heavily on this resource.[22]

Consider Extended Example I (end of the paper), a conversation among four participants which continued for about two days. For clarity, everyone involved in these conversation is given a single letter name; I am Dr. S, JP Massar is Mr. M, Jeff Elhai is Dr. E. All other participants are similarly anonymized. Recall that Mr. M and I are computer

[22] That I rely on email may seem odd considering my claim that the trading zone surrounds the BioBike web-based tool. Really the trading zone comprises the whole set of electronic communications between the participants, although BioBike programs are nearly always the focus of discussion, and it is within BioBike that programs – the interlingual lexicon – exist. Very often there is BioBike+email or BioBike+phone interaction between the participants. They can even collaborate simultaneously on the BioBike server itself.

scientists (although I'm also a biologist, but to a lesser extent than Dr. E.), and that Dr. E is a biologist with some experience in programming.

The conversation begins with a technical question from a biologist-user (DR. X) in Michigan [Oct 21, 2004 7:09 AM]. He is trying to find housekeeping genes and having some programming trouble. I answer him [Oct 21, 2004 8:25 AM] with what turns out to be an incorrect answer. (I forgot that the genomes of bacteria are circular!) Next, Dr. E. answers [Oct 21, 2004 7:53 AM]. At Oct 21, 2004 9:34 AM Mr. M writes to Dr. E with several points: First he asks what housekeeping genes are, and then proceeds to give programming tips to Dr. E. Immediately thereafter Mr. M writes to me [Oct 21, 2004 9:43 AM] explaining why my solution was wrong, which is confirmed slightly later in a message from Dr. E [Oct 21, 2004 10:50 AM]). At Oct 21, 2004 10:07 AM Dr. E writes back to Mr. M., explaining what housekeeping genes are. Ensues thereafter a long conversation between Mr. M and Dr. E regarding the best ways to program certain aspects of this problem, details of BioLisp and BioLite programming are discussed, and the conversation ranges into the philosophy of the BioLingua and BioLite projects (e.g., Dr. E: "I hope you're suggesting these methods for my education. MAPCARNN and LAMBDA will certainly blow first-time users away! (MAPCARNN blew ME away. I had to go look it up, thereby finding a new useful function.)" Mr M.: "Yes, and as illustrations of different ways of doing things that users may wish to consider." DR. X finally gets his detailed solution at Oct 22, 2004 11:13 AM (from Mr. M), although the conversation between Mr. M and Dr. E regarding destructive functions continues for a bit longer.

There are many remarkable features of this interaction. First, rather than simply answer Dr. X's question, Mr. M and Dr. E engaged in a complex discussion ranging from how to solve the specific problem, to the philosophy of how biologists should be taught to program. In the process, they taught one another about their respective ways of thinking and doing things. Mr. M and Dr. E are not talking either purely biology nor purely software engineering, but have developed a specific pidgin, BioLisp (or its simpler cousin, BioLite) within which they discuss ways of doing what it is that needs to be done.

Each is, as well, learning the other's specific meanings (e.g., what housekeeping genes are; the difference between a "list itself" and a "variable pointing to a list" [Mr. M to Dr. E, Oct 22, 2004 10:10 AM]).

I wish to claim that this specific sort of collaboration is afforded by the special situation and properties of the Lisp-based BioBike. BioBike and its base language, Lisp, are especially suited to the development of technical pidgins. As evidence, note for example these lines from the message from Mr. M to Dr. X on Oct 22, 2004 11:13 AM:

```
<3>> (defun is-a7120-gene (g) (equal a7120 (#^Organism g)))
:: IS-A7120-GENE

<4>> (setq housekeeping-genes (#^Go.Related-Genes #$Go.Metabolism))
:: (#$A7120.alr7635 #$A7120.alr7622 #$A7120.all7592 #$A7120.alr7073
    ...)

<5>> (setq a7120-housekeeping-genes
           (remove-if-not 'is-a7120-gene housekeeping-genes))
```

Here we are witnessing the development of a new noun (variable): "housekeeping-genes" (line 4), and a new verb (function): "is-a7120-gene" (line 2)! Finally, in line 5, we apply the new verb to the new noun, thereby computing Dr. X's desired result (an saving it in yet another new noun: `a7120-housekeeping-genes`). These nouns and verbs exist neither in the biological nor in the computational domains—they are specific to the field of computational genomics—and once they have been defined by one member of the BioBike community they can be used by any member of that community. Indeed, recall that Lisp begat BioLisp begat BioLite, each a pidgin of the previous language by specific virtue of adding new vocabulary from the new computational genomics domain; the pidgin is precisely not fixed! The members of the community of BioBike users continue to extend and modify it each time they create new functions (verbs) or variables (nouns), or modify existing ones. In order for this to happen, the users, whether they be biologists or software engineers, need to obtain interactional expertise in one another's domain through interactions such as the one we just examined. This interactional expertise

enables those collaborations to continue and expand. BioBike is built expressely for this purpose.[23]

The second and third examples (end of the paper) provide additional evidence of the co-evolution of interactional expertise by the BioBike community along with development of the BioBike pidgins (BioLisp and BioLite). Here, again, software engineers (Mr. M, and in the second and third examples, Dr. W as well) learn more biology at the same time that biologists (primarily Dr. E, also Dr. X) learn more computing, while at the same time the pidgin that is the BioBike biocomputing platform evolves.

In Example II the biological concept of "orthology" is considered and compared with the BioLisp/BioLite instantiations of that concept. Whereas real world concepts—even those defined by scientists—can be fuzzy and ambiguous, computer programs must have specific definitions. Dr. W (a senior computer scientist working with me on related biocomputing problems) needs to know certain properties of orthology, but even within biology there are many meanings of orthology, and only one specific meaning in the BioBike pidgin. This negotiation of meaning emphasizes the complex nature of scientific computation in general: we have (i) the real world, (ii) scientists' understandings of the real world, (iii) a specific computational implementation of the scientists' understandings, and (iv), other engineers' and scientists' understandings of the computational implementation.

In the third example we observe a much more complex interaction of a similar nature. Here Dr. W is trying to develop a computational implementation (within BioBike) of a problem related to plasmodium, posed by a biologist (Dr. Y), but Dr. W does not understand some of the fundamental aspects of the problem. Once Dr. S confirms that Dr. W has the correct model, Dr. W proceeds with a trial formulation. The conversation

[23] Although, of course, we did not think of this in terms of trading zones and interactional expertise at the time.

proceeds through several interactions (elided here), and after it is concluded, Dr. W asks: "What sort of thing is a plasmodium anyhow?"[24]

6. What sort of thing is a BioBike anyhow?

Collins and Evans (2007) describe interactional expertise—the mastery of the language of a domain—as "the medium of interchange within large scale science projects, where not everyone can be a contributor to everyone else's narrow specialism." They propose that such expertise is gained with more and more discussion of the science: "[W]here interactional expertise is being acquired there will be a progression from `interview' to `discussion' to `conversation' as more and more of the science is understood. There is no sudden `aha moment' which marks the switch to mastery of interactional expertise but its steady acquisition can nevertheless be recognised." One would expect, therefore, to find, at the very least, interactional expertise in what Galison calls a scientific trading zone. Is BioBike such a trading zone, and are we watching the participants develop interactional expertise?

They are developing this and more. As the BioBike project progress, an explicit set of new shared concepts is being developed upon which the interaction is focused. These concepts are computer programs (BioBike functions) and are at the same time boundary objects and form the dictionary of the collaborative communication. BioBike does seem to satisfy the requirements of being an (online) trading zone and the BioLisp/BioLite functions serve the triple role of subject of the trade, boundary objects in the interaction, and the pidgin that is formed as the trade progresses.

In all of these examples biologists and computer scientists are co-evolving a pidgin which exists on two planes: first in their conversation, and second in the biocomputing platform that they are developing together and which is being used by the biologists. The facility to dynamically extend the system's working vocabulary makes BioBike unique among computationally-based collaboration tools which, although they often support

[24] Plasmodium is the organism that causes malaria.

conversations among participants, do not usually themselves grow organically through such interactions.[25] Indeed, it is tempting to think of BioBike not so much as a tool but as a participant in the conversation.

In developing BioBike, the biologists and computer scientists are developing a fundamental biological instrument—a biocomputational tool that must be used, and indeed is being used—by biologists to get real scientific work done—work that they could not get done any other way. In the record of interactions between the participants in the BioBike community we have a window onto the details of the evolution of this important instrument, and what we see is extremely complex. Not merely learning to talk to one another, the scientists, engineers, and BioBike are doing real work of biocomputation and at the same time as they are evolving the way that this work gets done, they are extending their own understandings, amoeba-like into one another's areas of expertise. From the point of view of a given individual, Piaget's concept of adaptation, comprised of the twin processes of assimilation and accommodation, seems an apt description: As new concepts are encountered, the individual assimilates them into his/her/its understanding, which will almost always require change (accommodation) of its/her/his understanding. But from the point of view of the community, the process is internal: we are seeing the community's internal accommodation process wherein participants' knowledge and skills—here including the "skills" (i.e., functionality) of the BioBike platform itself—are slowly becoming more and more intercalated.

Specialized programming platforms are becoming increasingly important as computers infuse greater parts of our daily lives and as we wish to have greater control over them. I have argued that the programming languages that are the heart of computing platforms serve as, at the same time, inter-languages in the trading zones that are these platforms,

[25] To some extent this is a property simply of the current domain – being computational work. Of course a collaboration tool that enables geologists, for example, to talk among themselves would not be expected to grow new sorts pick axes. Yet as computers come to be more and more infused into all aspects of science, we might expect the development of tools such as BioBike to become more and more important; and it's most important feature is, I believe, its technical extensibility.

and that the functions and objects of those languages serve as boundary objects in these trading zones. I have also argued that certain types of programming languages, esp. highly extensible and highly interactive languages like Lisp, are well suited to playing this dual role. In the BioBike world not only do the participants in the collaboration become partial (at least interactional) experts in one another's domains, they also co-evolve the BioBike inter-languages themselves—BioLisp and BioLite. As BioBike-like computational communities of practice appear throughout science they may afford excellent opportunities to study this interesting sort of co-evolution.

7. Appendix 1: Example I: Oct 21-22, 2004

[The participants in this conversation are in three different US time zones (all times are in Pacific time). Irrelevant names of users are X'ed out and, where necessary, assigned number (e.g, X1, X2, in order to disambiguate them). Irrelevant parts of messages have been silently deleted, and some minor typographical changes have been made in the interest of readability without loss of relevant content regarding the present discussion.]

From: DR. X <[Michigan]>
Date: Oct 21, 2004 7:09 AM
Subject: Help with BioLingua

I'm a new user of BioLingua, with very little experience in computer programming. I'm searching for housekeeping genes in Anabaena 7120 that are longer than 3000 bp. I could load Anabaena sequences by:
>> (setf an (load-organism "A7120"))

I found genes that are involved in metabolism by:
>> (setf metabolism (find-frames "metabolism"))

I got a list of related genes by:
>> (df #$go.metabolism)

now I want to find the length of each gene in the list "metabolism" and check if it is longer than 3000. This is where I don't know what function to use.

I tried to start with the loop:

```
(LOOP FOR LongSequences in (GENES-OF a7120)
   as length = (LENGTHS-OF LongSequences)
   when (length > 3000)
Collect LongSequences)
```

or some variation of it. None worked although I'm sure I'm pretty close.

I also do not understand why I didn't get a list of genes when I used the "find-frames" command (function?), what exactly the value of this command?

From: Dr. S <[California]>
To: DR. X <[Michigan]>
Date: Oct 21, 2004 8:25 AM
Subject: Re: Help with BioLingua

Thanks for asking us about this. You're right that you're very close. In "raw" BioLingua, the query you might use is:

```
(setf an (load-organism "A7120"))

(loop for gene in (#^genes an)
     when (> (abs (- (#^from gene) (#^to gene)))  3000)
     collect gene)
```

[Returns 176 genes.]

Or, slightly slower (because it has to get the sequence of each gene):

```
(loop for gene in (#^genes an)
     when (> (length (extract-sequence gene)) 3000)
     collect gene)
```

[Returns 172 genes. Hmmmmm. I'm not sure why they are different. Possibly there are some where the stop codon is right at the end, or there might be errors in the database that we downloaded for Anabaena.]

Dr. E can give you the BioLite versions of these, which are slightly simpler.

Now, to get only those involved in metabolism, you can do this slightly more complex expression:

```
loop for gene in (#^genes an)
     when (and (> (abs (- (#^from gene) (#^to gene))) 3000)
```

```
                (member #$go.metabolism (#^go-id gene)))
     collect gene)
```

And end up with 17 genes.

On your other questions:

> I also do not understand why I didn't get a list of genes when I
> used the "find-frames" command (function?), what exactly the value
> of this command?

Ah. This actually gives you the FRAME representing the CONCEPT of metabolism (from the Gene Ontology). The genes themselves are in the slot called #^go.related-genes of that frame. So, another way to approximate the above loops is:

```
 (loop for gene in (#^go.related-genes #$go.metabolism)
     when (> (abs (- (#^from gene) (#^to gene))) 3000)
     collect gene)
```

Unfortunately, this includes several synechocystis genes, which you could either sort out by hand, or use a slightly more complex query to find.

ps. Dr. E will probably chime in with other answers to your question that conform a little more to the BioLite way of programming. It's nearly the same, but some of the function names are slightly different, and the syntax is slightly simpler.

From: Dr. E <[Virginia]>
To: DR. X <[Michigan]>
Date: Oct 21, 2004 7:53 AM
Subject: Re: Help with BioLingua

It's remarkable that you got as far as you have!

Here's one way to get a list of genes that you can then sift through by length:

```
  (LOOP FOR frame IN (FIND-FRAMES "metabolism")
        AS genes = (GET-ELEMENT GO.related-genes FROM frame)
      WHEN (EXISTS genes)
    APPEND genes)
```

If you like what you get, you can save the result in a variable:

```
  (ASSIGN metabolic-genes *)
```

The asterisk inserts the results of the previous operation). To find out how many genes you got:

```
  (LENGTH metabolic-genes)
```

If you're concerned about duplicates, you can get rid of them:

```
  (DELETE-DUPLICATES metabolic-genes)
```

Now you can go through picking out the big genes:

```
  (LOOP FOR gene IN metabolic-genes
        AS length = (LENGTH-OF gene)
      WHEN (> length 3000)
    COLLECT gene)
```

What you did could be made to work with minor modification:

```
  (LOOP FOR gene IN (GENES-OF a7120)
        AS length = (LENGTH-OF gene)
      WHEN (> length 3000)
    COLLECT gene)
```

Note that (1) the loop deals with one gene at a time and (2) BioLingua changes the usual order of operands (greater-than must come first.

Now,

```
   (INTERSECTION * metabolic-genes)
```

gives you those Anabaena genes with lengths greater than 3000 that are also in the set of metabolic genes.

From: Mr. M <[California]>
To: Dr. E <[Virginia]>
Date: Oct 21, 2004 9:34 AM
Subject: Re: Help with BioLingua

What are 'housekeeping' genes?

```
>    (LOOP FOR frame IN (FIND-FRAMES "metabolism")
>            AS genes = (GET-ELEMENT GO.related-genes FROM frame)
>         WHEN (EXISTS genes)
>       APPEND genes)
```

But this does not restrict the genes to the Anabaena 7120 organism.

You could do
APPEND (remove-if-not 'is-anabaena7120-gene genes)
and
(defun is-anabaena7120-gene (gene) (eq ana7120 (#^Organism gene)))

```
>If you're concerned about duplicates, you can get rid of them:
>
>    (DELETE-DUPLICATES metabolic-genes)
```

Ruh roh. This is faulty. You generally need to reassign metabolic-genes to the result of DELETE-DUPLICATES.

```
>Now you can go through picking out the big genes:
>
>    (LOOP FOR gene IN metabolic-genes
>            AS length = (LENGTH-OF gene)
>         WHEN (> length 3000)
>      COLLECT gene)
```

Or

```
(remove-if-not (lambda (gene) (> (length-of gene) 3000))
   metabolic-genes)
```

```
>Or better (since a big list of genes will probably not be very
>edifying):
>
>    (LOOP FOR gene IN metabolic-genes
>            AS length = (LENGTH-OF gene)
>            AS description = (DESCRIPTION-OF gene)
>         WHEN (> length 3000)
>      COLLECT (LIST gene length description))
```

Or

```
(mapcarnn
  (lambda (gene)
     (let ((len (length-of gene)))
      (when (> len3000) (list gene len (description-of gene)))))
   metabolic-genes))
```

```
>Note that (1) the loop deals with one gene at a time and (2) BioLingua
>changes the usual order of operands (greater-than must come first.
```

Better to emphasize that the function ALWAYS comes first. We could emphasize this more in our introductory tutorials.

```
>gives you those Anabaena genes with lengths greater than 3000 that are
>also in the set of metabolic genes.
```

Again you probably want to assign the result to some variable.

From: Mr. M <[California]>
To: Dr. S <[California]>
Date: Oct 21, 2004 9:43 AM
Subject: Re: Help with BioLingua

...
>[Returns 176 genes.]
...
>[Returns 172 genes. Hmmmmm. I'm not sure why they are different.
> Possiblythere are some where the stop codon is right at the end,
> or there might be errors in the database that we downloaded
> for Anabaena.]

There are at least two complications:

The lack or inclusion of '*' in the sequence.

Whether any of the contigs are circular and have a gene or genes that cut across the arbitrary 0 (i.e. 'wrap').

The A7120 organism has multiple circular contigs.

From: Dr. E <[Virginia]>
To: Mr. M <[California]>
Date: Oct 21, 2004 10:07 AM
Subject: Re: Help with BioLingua

>What are 'housekeeping' genes?

Housekeeping genes are those genes that are useful for the general maintenance of the cell under normal conditions. The term is usually used in the context "just houskeeping genes", implying "not interesting". But for those looking metabolism as a whole, they can be very interesting.

...
>But this does not restrict the genes to the Anabaena 7120 organism.

Unfortunately it almost does. Only A7120 and S6803 have genes in GO frames. But at some point you DO have to get rid of the S6803 genes.

>You could do
>

> APPEND (remove-if-not 'is-anabaena7120-gene genes)
> and
> (defun is-anabaena7120-gene (gene) (eq ana7120 (#^Organism gene)))

My approach is to provide suggestions using only tools that the user is likely to use over and over again. In that light, the following, I think, is more accessible:

 (INTERSECTION * (GENES-OF A7120))

>>   (DELETE-DUPLICATES metabolic-genes)
>
>Ruh roh. This is faulty. You generally need to reassign metabolic-genes
>to the result of DELETE-DUPLICATES.

DELETE-DUPLICATES is a destructive function.
REMOVE-DUPLICATES is nondestructive.

[...]

I hope you're suggesting these methods for my education. MAPCARNN and LAMBDA will certainly blow first-time users away! (MAPCARNN blew ME away. I had to go look it up, thereby finding a new useful function).

Thanks,

Jeff E

From: Mr. M <[California]>
To: Dr. E <[Virginia]>
Date: Oct 21, 2004 10:09 AM
Subject: Re: Help with BioLingua

[...]
>DELETE-DUPLICATES is a destructive function.

Yes it is. ***> BUT YOU STILL NEED TO ASSIGN THE RESULT !!!!!! <***

You are asking for serious trouble if you do not.

>I hope you're suggesting these methods for my education.

Yes, and as illustrations of different ways of doing things that users may wish to consider.

From: Dr. E <[Virginia]>
To: Mr. M <[California]>
Date: Oct 21, 2004 10:50 AM
Subject: Re: Help with BioLingua

>Whether any of the contigs are circular and have a gene or genes that
>cut across the arbitrary 0 (i.e. 'wrap').

FOR $100 AND THE GAME!

The genes found by the first method but not the second are actually shorter than 3000 nucleotides, but they all cross the boundary of a replicon. That makes the FROM field larger (MUCH larger) than the TO field. EXTRACT-SEQUENCE is clever enough to figure out the truth.

>>DELETE-DUPLICATES is a destructive function.

>Yes it is. ***> BUT YOU STILL NEED TO ASSIGN THE RESULT !!!!!! <***
>You are asking for serious trouble if you do not.

What serious trouble am I asking for?
Answer after some experimentation:
 ... that the variable really isn't modified.

DELETE-DUPLICATES does not modify lists (or strings)
>>(SETF x '(1 2 3 4 5 4 3 2 1))
:: (1 2 3 4 5 4 3 2 1)
>> (DELETE-DUPLICATES x)

```
:: (5 4 3 2 1)
>> x
:: (1 2 3 4 5 4 3 2 1)

DELETE-DUPLICATES *does* modify arrays
>> (SETF x #(1 2 3 4 5 4 3 2 1))
:: #(1 2 3 4 5 4 3 2 1)
>> (DELETE-DUPLICATES  x)
:: #(5 4 3 2 1)
>> x
:: #(5 4 3 2 1)
```

"delete-duplicates is like remove-duplicates, but delete-duplicates may modify sequence." "delete-duplicates, when sequence is a list, is permitted to setf any part, car or cdr, of the top-level list structure in that sequence."
   - Common Lisp HyperSpec

It sounds like that DELETE-DUPLICATES *sometimes* modifies a list, but what use is a destructive function that doesn't reliably destroy? Why didn't the Lisp Gods make the function ALWAYS replace the list argument?

Is there a rule governing when "may modify" comes into play?

From: Mr. M <[California]>
To: Dr. E <[Virginia]>
Date: Oct 21, 2004 9:48 PM
Subject: Re: Help with BioLingua

Someone (presumably the original responder) needs to contact the original poster and explain that the solutions using FROM and TO in such a naive way fail miserably because of circular contigs.

From: Mr. M <[California]>
To: Dr. E <[Virginia]>
Date: Oct 22, 2004 10:10 AM
Subject: Re: Help with BioLingua

>Is there a rule governing when "may modify" comes into play?

You are confusing the list itself with the variable that points to the list.

Destructive functions modify the list they are given, they don't modify the variable that evaluates to the list (they couldn't possibly, since they are functions, not macros)

I.e.
(SETQ x '(1 2 3))
(DELETE 1 X)

the delete function knows nothing about 'X' itself. It just gets the value of X, which is the list '(1 2 3).

That's why to change what X points at, you must do
(SETQ X (DELETE 1 X))

A destructive function may actually modify the list it is given, or it may not.

Suppose I delete the number 1 from the list (1 2 3)

All I have to do to achieve this is return a pointer to the sublist (2 3).

But suppose I delete the number 2 from the list (1 2 3)

Now I have to physically go in and 'unsplice' 2 from the list.

This fact is the reason why the following is true:

```
CG-USER(11): (setq x '(1 2 3))
(1 2 3)
CG-USER(12): (delete 1 x)
(2 3)
```

```
CG-USER(13): x
(1 2 3)
CG-USER(14): (setq y '(1 2 3))
(1 2 3)
CG-USER(15): (delete 2 y)
(1 3)
CG-USER(16): y
(1 3)
CG-USER(17):
```

In order to understand this further you would need to understand 'box notation' which explains how lists are represented in computer memory.

From: Mr. M <[California]>
To: DR. X <[Michigan]>
Date: Oct 22, 2004 11:13 AM
Subject: Re: Help with BioLingua

Here's an abbreviated script showing how to do exactly what you want, starting from after you found the GO.METABOLISM frame.

Hope this helps.

[...]

```
<2>> a7120
:: #$anabaena_pcc7120

<3>> (defun is-a7120-gene (g) (equal a7120 (#^Organism g)))
:: IS-A7120-GENE

<4>> (setq housekeeping-genes (#^Go.Related-Genes #$Go.Metabolism))
:: (#$A7120.alr7635 #$A7120.alr7622 #$A7120.all7592 #$A7120.alr7073
 #$A7120.alr5353 #$A7120.alr5286 #$A7120.alr5200 #$A7120.alr5182
 #$A7120.all5094 #$A7120.all5022 #$A7120.alr4976 #$A7120.alr4944
 #$A7120.alr2505 #$A7120.alr2495 #$A7120.all2347 #$A7120.all2166 ...)
```

```
<5>> (setq a7120-housekeeping-genes (remove-if-not 'is-a7120-gene
housekeeping-genes))
:: (#$A7120.alr7635 #$A7120.alr7622 #$A7120.all7592 #$A7120.alr7073
 #$A7120.alr5353 #$A7120.alr5286 #$A7120.alr5200 #$A7120.alr5182
 #$A7120.all2644 #$A7120.all2643 #$A7120.all2642 #$A7120.all2635
 #$A7120.alr2505 #$A7120.alr2495 #$A7120.all2347 #$A7120.all2166 ...)

<6>> (length housekeeping-genes)
:: 229

<7>> (length a7120-housekeeping-genes)
:: 144

<8>> (setq result (loop for g in a7120-housekeeping-genes
           when (> (length (extract-sequence g)) 3000)
           collect g))
:: (#$A7120.alr3809 #$A7120.alr2680 #$A7120.alr2679 #$A7120.alr2678
 #$A7120.all2649 #$A7120.all2648 #$A7120.all2647 #$A7120.all2646
 #$A7120.all2645 #$A7120.all2644 #$A7120.all2643 #$A7120.all2642
 #$A7120.all2635 #$A7120.all1695 #$A7120.all1649 #$A7120.all1648
 #$A7120.all1643)

<9>> (length result)
:: 17
```

From: Dr. E <[Virginia]>
To: Mr. M <[California]>
Cc: BioLinguaSupport@lists.stanford.edu
Date: Oct 22, 2004 12:24 PM
Subject: Re: Help with BioLingua

Hello JP,

>You are confusing the list itself with the variable that points to the list.

Am not!

```
>Destructive functions modify the list they are given, they
> don't modify the variable that evaluates to the list
> (they couldn't possibly, since they are functions, not macros)

(DEFUN Real-delete-duplicates (entity)
"
 - Destructively removes duplicates from list or array
 - Destructively removes duplicates from string but, unfortunately
    does not change the length of the string from the original
"
 (LET ((new-entity (REMOVE-DUPLICATES entity)))
   (COND
      ((LISTP entity)
         (SETF (CAR entity) (CAR new-entity))
         (SETF (CDR entity) (CDR new-entity))
         new-entity)
      ((STRINGP entity)
         (SETF (SUBSEQ entity 0) (SUBSEQ new-entity 0)))
      ((VECTORP entity) new-entity)
      (T (ERROR "Inappropriate type in REAL-DELETE-DUPLICATES")))))

>> (SETF x '(1 2 3 4 5 4 3 2 1 3 5 3 1 7))
:: (1 2 3 4 5 4 3 2 1 3 5 3 1 7)
>> (REAL-DELETE-DUPLICATES x)
:: (4 2 5 3 1 7)
>> x
:: (4 2 5 3 1 7)

>> (SETF x "1 3 5 3 1 3 5 7 9")
:: "1 3 5 3 1 3 5 7 9"
>> (REAL-DELETE-DUPLICATES x)
:: "1357 9"
>> x
:: "1357 93 1 3 5 7 9"
         ^^^^^^^^^^^
```

(I don't know how to get rid of these unwanted characters in a function)

The question remains: if DELETE-DUPLICATES does not reliably do anything different from REMOVE-DUPLICATES, then why does it exist?

From: Mr. M <[California]>
To: Dr. E <[Virginia]>
Cc: BioLinguaSupport@lists.stanford.edu
Date: Oct 22, 2004 12:43 PM
Subject: Re: Help with BioLingua

REMOVE-DUPLICATES is equivalent to

1. Make a copy of the list or vector
2. Call DELETE-DUPLICATES on the copy and return what DELETE-DUPLICATES returns.

```
(SETQ X '(1 2 2 3 3))
(REMOVE-DUPLICATES X)
(1 2 3)
X
(1 2 2 3 3)
(SETQ Y '(1 2 2 3 3))
(DELETE-DUPLICATES Y)
(1 2 3)
Y
(1 2 3)
```

In other words, REMOVE* functions never change anything about the data they are passed; they create new data, then modify it.

DELETE* functions have carte blanche to change anything about the data they are passed, but only do so if they need to and can do so.

```
(SETQ X "aabbcc")
(DELETE-DUPLICATES X)
"abc"
X
"aabbcc"
```

In this case, in this implementation of Lisp, there is no way to shorten the length of a string. Since the Lisp cannot smash the string to make it shorter, it is allowed to return a different string.

But consider:
(setq x #(1 1 2 2 3 3))
(DELETE-DUPLICATES X)
#(1 2 3)
X
#(1 2 3)

In this case, in this implementation of Lisp, the implementation allows the vector to be shortened 'in place'. So the original vector itself is smashed.

The purpose of DELETE* functions is to avoid making a copy when possible.

From: Mr. M <[California]>
To: Dr. E <[Virginia]>
Cc: biolinguasupport@lists.stanford.edu
Date: Oct 23, 2004 6:07 PM
Subject: Re: Help with BioLingua

Just a side note that using INTERSECTION here as is being done instead of something like REMOVE-IF-NOT as above is much less computationally efficient.

(Which doesn't matter in this case, but certainly would if something like this were done in the inner loop of a program)

8. Appendix 2: Example II (abbreviated)

Dr. W: “when i look for genes in s6803 that have orthologs that are also in s6803, i only get genes that are orthologous to themselves. Are there distinct genes that are both in s6803 and are orthologs of each other?”

Dr. E: “Technically, the question makes no sense. Genes are orthologs of their evolutionary line of descent matches that of their organisms. It doesn't make sense to ask what is an ortholog of a gene in the same organism. But you probably intend "ortholog" to mean "similar gene", so to hell with semantics. The function (ORTHOLOG-OF gene IN set-of-organisms) returns at most one gene per organism screened. I don't know about the function you used. You could use (GENES-SIMILAR-TO gene IN set-of-organisms) which will return ALL genes sufficiently similar to the given gene. "sufficiently similar" can be adjusted if you like.”

Dr. W: “Is "ortholog" a symmetric relation? i mean, if A is an ortholog of B, is B also said to be an ortholog of A? Also, is a gene said to be an ortholog of itself?”

Dr. S: “There are various sense of ortholog. What it *theoretically* means is that the genes derived from the same parent evolutionarily, and/or that they serve the same function. What it means *practically* is: Genes in different organisms which are one another's best match, where "match" can be variously defined, but usually just means: Best blast hit.”

Dr. E: “The most common working definition of "ortholog" (and the one we use) is: gene X in organism A is orthologous to gene Y in organism B if and only if gene Y is the gene most similar in organism B to gene X and gene X is the gene most similar in organism A to gene Y and the degree of similarity exceeds a certain threshold Given this definition (and there are others), orthology of A to B implies orthology of B to A.”

Dr. W: “and A is an ortholog of itself. But it is possible that A is an ortholog of B and B is an ortholog of C but A is not an ortholog of C, because the threshold is exceeded; the relationship is not transitive. Just as well---the axiom for transitivity has a bad effect on the search space for the theorem prover.”

Mr. M: “Not complete. Try this: It is possible that a in A is an ortholog of b in B, and b in B is an ortholog of c in C, but a in A is not an ortholog of c in C, because EITHER the threshold is exceeded, OR there exists a c' in C such that a in A is orthologous to c' in C, and c /= c'. For instance, if c in C is identical in sequence to b in B, and c' in C is identical in sequence to a in A, the latter condition will hold.”

Dr. W: “Right. I hadn't noticed that "closest" condition. In fact, it could be that a in A is othologous to b in B, but b in B is not orthologous to a in A, because there is some gene a' in A that is closer to b than a is. So the relation is not even symmetric.”

Dr. E: “No, if there were some gene a' in A that is closer to b than a is, then a would not be orthologogous to b. The relation IS symmetric, but it is not transitive.”

9. Appendix 3: Example III (abbreviated)

Dr. W: here are some questions that Dr. Y formulated.

[quoted from Dr. Y] “Q1: What genes are expressed highest before invasion of Red blood cells by plasmodium falciparum? First we would need to identify the protein-members involved in the process of "invading Red blood cells". Some of these members will be plasmodia proteins and some will be human proteins. We will have microarray data about gene expression in plasmodia at time points starting before the process of invasion begins and during the process of invasion as well. We would have to "match-up" the steps in invasion with the time line of microarray data collection, identify the experiment-sample that is before the critical invasion step, find the genes that have highest expression and report those.”

[Dr. W, continues] are the genes in the red blood cell or in the plasmodium? how do you tell how much a gene is expressed? is there a number that measures this? in what sense is a protein-member involved in the process? are the proteins part of the red blood cell or part of the plasmodium? etc.

Dr. S: This sounds much more complex than it is. They have a microarray with a bunch of genes, some from blood cells, some from plasmodium, and a time series of these. All they want to know is which genes are at their peak before, during and after the invasion.

Dr. W: here's what is was thinking of for this one: we would introduce the relation gene-in-process(?gene, ?process) which holds if ?gene plays a role in ?process, and the function gene-expression(?gene, ?time-point), which accesses the microarray data to say to what degree ?gene is expressed at ?time-point (the value is a number. what unit?) [...] Then the query: [...]will find a gene involved in the invasion, a time-point that is before the invasion, and the expression of the gene at that time-point. [...] Does my formulation seem reasonable? Are there better names than the ones i invented (gene-in-process, gene-expression, plasmodium-invasion-1, etc.? This will work but might take a long time---do you know of more sophisticated ways of searching the data to find a maximum?

Dr. W: [again quoted from Dr. Y] “Q2: If I block activity of protein X, will invasion not occur? First we would need to identify the protein-members involved in the process of "invading Red blood cells". Then retrieve their annotations (to find out what their functions are)...then for the protein specified (X), try to find alternative paths in the invasion process that can still reach the end state of entring the Red blood cell but do not require the functioning of the protein X.”

there is something i am confused about here----i think of the invasion as being a single path, but the paragraph indicates that it consists of many paths, some of which succeed in entering the red blood cell

and some do not. when i am talking about a particular invasion for which we have microarray data, what does it mean to say that some paths are executed and some are not, but could have been? i can make sense of it by thinking there is an environment in which there are many possible execution paths (like a nondeterministic program). one of these paths has actually occurred, but others are possible.

Dr. S: Yes, this is the correct model.

Dr. W: in phrasing a counterfactual problem, i am considering the same environment (initial state) and supposing that i am blocking certain of the possible paths. i am asking if any of those paths could be a successful invasion. so i would formulate the question (if i have rephrased it correctly) as follows. execute(?path, ?state0, ?state1) is an execution of a path which begins in ?state0 and ends in ?state1. path-requires-protein(?path, ?protein) holds if ?path requires the activity of ?protein. red-blood-cell-invaded(?state) is true if the red blood cell in question is invaded in ?state. Let proteinX be the protein in question. We assume that state0 is an initial state, state1 is a final state, and path01 is a path such that execute(path01, state0, state1) and red-blood-cell-invaded (state1). [...]

[...]

Dr. W: what sort of a thing is a plasmodium anyhow? is it an organsim?

References

Collins, H. & Evans, R. (2002). The Third Wave of Science Studies: Studies of Expertise and Experience. Social Studies of Science, 32, 235-296.

Collins H. and Evans, R, (2007). Rethinking Expertise Chicago: University of Chicago Press

Crease, R.P. (2006). From workbench to cyberstage. In E. Selinger (Ed.), Postphenomenology: A critical companion to Ihde. Albany: State University of New York Press.

Curtis, P., & Nicols, D. (1992). Mudding: Social Phenomena in Text-Based Virtual Realities. Proceedings of the Conference on Directions and Implications of Advanced Computing (DIAC-92). Berkeley, Calif. (Also available as Xerox PARC technical report CSL-92-4.)

Galison, P. (1997). Image and Logic. Chicago: University of Chicago Press.

Massar, J.P., Travers, M., Elhai, J., & Shrager, J. (2005). BioLingua: A programmable knowledge environment for biologists. Bioinformatics, 21, 199-207.

Sahdra, B., & Thagard, P. (2003). Procedural knowledge in molecular biology. Philosophical Psychology, 16, 477-498.

Shrager, J. (2000). Diary of an Insane Cell Mechanic. http://nostoc.stanford.edu/jeff/personal/diary/diary.html (last accessed 20060904).

Shrager, J. (2004). On being and becoming a molecular biologist: Notes from the Diary of an Insane Cell Mechanic. In M. Gorman, R. Tweney, D. Gooding, & A. Kincannon (Eds.), New Directions for the Cognitive Study of Scientific and Technological Thinking. Hillsdale, NJ: Lawrence Erlbaum Associates.