StructProc8b:
A Language for Scientific Reasoning

Joseph Phillips, Donald Bartoli, Michael Cohen

Applied Philosophy of Science
4330 S. Michigan Ave.
Chicago, IL 60653, USA
jphillips@scienceomatic.com, penguinkin@gmail.com, mike@scienceomatic.com

Abstract

Scientific knowledge representation and reasoning presents unique challenges which the computer language community under-appreciates. We present early work on a scientific reasoning environment and knowledge base designed to simultaneously support a community of scientists, to handle scientific knowledge like floating point distributions, meta-data and meta-knowledge, and to treat explanations as natural objects. We present the results of preliminary experiments on the speed with which it handles meta-data.

1. Introduction

Science is different things at different scales. Macroscopically, science is a multi-generational, transnational, multicultural human undertaking that seeks to obtain ever more observations of the Universe. It seeks to explain these observations in as metaphysically-neutral a manner as deemed warranted. At medium scales, science is what a community of scientists does. Scientific communities share interest in a common field and hold broad yet mutable principles of how to accept and reject observations and explanations. More specific still, science is the day-to-day actions of the scientist going about their business: observing, calculating, theorizing, collaborating, reading, and writing.

As hopefully non-controversial as the previous paragraph was, it is easy to overlook, for that is not how science is studied. Indeed, the fragmentation of disciplines which study science (history, philosophy, sociology) echoes the fragmentation of science itself into fields, sub-fields, and Kuhnian “paradigms” within sub-fields (Kuhn 1962).

Unfortunately, attempts at applying computing to science have been not only similarly fragmented, but also piecemeal. Large computing projects have been undertaken in science, the Human Genome Project (Frenkel 1991) and SETI@home (Korpela et al. 2001) being two ready examples, but they generally show immense computing power being applied to important, yet narrow problems.

A computing system designed to address the full sweep of science, by field and by scale, requires the following characteristics:

- Support for the truly large scale collaboration (*e.g.* support for a large portion of the world's scientists)
- Built-in support for handling scientific meta-data and meta-knowledge
- Built-in support for creating and manipulating explanations as first class objects in their own right
- Sophisticated support for numbers, especially distributions of floating points
- Allowance for detecting and limiting degrees of contradiction, and mechanisms for keeping it limited
- Allowance for building and supporting simulations (loops not being disfavored over recursion)
- Extendable support for incorporating both existing and new special-purpose computing packages

We are building the Scienceomatic 8b (SOM8b) as an environment for science: scientific reasoning, scientific representation, scientific visualization, etc. It is being designed to utilize existing, more specialized, programs to do reasoning, analysis, graphing and other computational tasks.

One of the main tools of SOM8b is its representational language: StructProc8b (SP8b). SP8b is designed to address all of the issues listed above. In this paper, we discuss its abilities to deal with explanations as objects and its ability to support a community of allied scientists. We concentrate on its ability to handle meta-data in a timely manner after discussing science and other approaches to representing scientific knowledge.

2. On Science and Meta-data

Science is a broad, dynamic and multifaceted endeavor. This section concentrates on three aspects, instead of attempting an in-depth survey.

First, at medium and large scales, science is a social enterprise: multi-generational, collaborative, and even tribal. New scientists are generally indoctrinated into their discipline by apprenticeships under more established scientists. Scientists of a given discipline often meet at conferences to share ideas and discuss their work.

Furthermore, scientists are expected to publish their work, except when outweighed by competitive concerns such as national security or trade secrecy. As scientific knowledge traverses the arc of documentation and publication – from lab notebook, to group meeting talk, to conference paper, to journal article, to textbook – the knowledge is cleansed, neatened and packaged. Tables of supporting data are sampled, summarized or altogether omitted. Data-points that do not fit well, and the rationalizations for why they do not belong, are removed. Counter arguments to conclusions are presented in outline form, if they are given at all.

Ideally, however, later compilations reference earlier sources so that the sufficiently curious and diligent can follow the links from the textbook back to the original lab notebook.

Second, scientific data and algorithms often have several unique characteristics. Scientists often deal with distributions of numbers, rather than individual floating point values. Scientific algorithms, especially numerically-intensive and simulation algorithms, heavily use variables and iterations.

Further, scientists place importance on meta-data and meta-knowledge associated with scientific values. Some data relates to the single numbers, like dimensions and units. Other data and knowledge are about organized collections of numbers. For example, when statistics are computed, it is expected that the sample size (*N*) will be given. Other data and knowledge still relate to multiple collections of data, multiple sets of experiments or a series of calculations. These include the answers to questions like "*Who did them?*", "*Which institution facilitated their research?*" and "*Who paid for the research?*"

Third, explanations are themselves scientific objects. In many ways, this situation extends mathematics' relationship with proofs. A proof is a mathematical object which is graded by at least two criteria: (1) "*Are all the premises reasonable and are all the steps valid?*" and (2) "*Is the whole structure elegant?*".

Purely deductive scientific reasoning steps, like *some* equation manipulations[1], share the same criteria. However, science is rarely purely deductive. The validity of non-deductive premises and steps are often gauged by fidelity to established protocol. In the experimental sciences, this means learning acceptable lab or field technique. As exciting research often pushes boundaries, new protocols have to be scrutinized relative to meta-protocols.

What counts as an explanation is specific to both the discipline and the Kuhnian paradigm of a scientific field. In some fields, diagrams are an essential part of explanations. Particular diagrams are highly idiosyncratic to particular disciplines (Giere 1999). Thought experiments are much the same. Further, what counts as an explanation depends on what one allows to count as reasonable alternatives (van Fraassen 1980). For example, in most sciences, constructing an explanation of why some natural phenomena exists in terms of its use to us (*e.g*. humanity) is not allowed. However, it is allowed in biology, in the narrow sense of advancing the adaptive fitness of the organism or species (Mayr 1988).

3. Prior Work

Computing has been applied to science and related technological problems since the invention of computers. However, then as now, the emphasis has been on solving specific scientific problems.

More recently people have been interested in scientific reasoning systems. In 2004, teams from SRI, Cycorp, Ontoprise Team and the Knowledge Based Systems Group at the University of Texas at Austin competed in building systems that could answer Advanced Placement chemistry questions (Friedland et al. 2004). One of these groups kept on going to build a system for biology too (Gunning et al. 2010). Their systems generally are able to handle pure college freshman-level science questions, but are brittle in that they are less able to reason about their own internal knowledge state.

Such systems advance the art of scientific knowledge representation, but as best we can tell, purposefully lack advanced abilities to handle conflicting data and knowledge. The fundamental assumption they seem to share is that science is a more-or-less static body of knowledge. Once entered correctly, the system is usable.

For us, however, science is an on-going collaborative process. Different people can hold different opinions at the same time. One person can hold different opinions at different times. We emphasize the history of changes of thought.

Bayesian network based systems are an alternative to logic and frame-based systems such as Cyc and the others mentioned above, and our own SP8b. We discuss why we think our frame-based system is better suited to open-ended scientific knowledge representation in section 6.

4. The Scienceomatic 8b and StructProc8b

The Scienceomatic 8b (SOM8b) is an environment for scientific reasoning designed to support a variety of general and specialized scientific programs, including theorem provers. We hope to grow it into an "operating system of scientific reasoning": an allocator of computing

1 An example of a non-deductive equation manipulation step comes from physics. The 2-body quantity known of the reduced mass of a system of two masses m_1 and m_2 is $(m_1*m_2)/(m_1+m_2)$. For $m_1 >> m_2$ it is common to approximate this just as m_2.

resources to various programs to solve given problems[2].

StructProc8b (SP8b) is a frame-based language for representing scientific knowledge. It is strongly-typed and classes are instances which may be created dynamically. SP8b has some inherent reasoning abilities, and will be a Turing-complete language in which reasoning algorithms may be written. However, currently its main use is to store knowledge for the SOM8b environment.

```
Idea
{*
  isA->assertZ(Idea);          // (1)
  instanceOf->assertZ(Idea);

  ideasIsImmutableA->          // (2a)
        subAssertZ(false);     // (2b)
  ideasIsSingletonA->
        subAssertZ(false);
  isIdeaANonspecifiedSetMemberA->
        subAssertZ(false);
*};
```

Figure 1. Explicit construction.

Figure 1 illustrates SP8b's explicit construction of `Idea`. (`Idea` is the root set of which everything else is an instance, akin to `Object` in Java.) When an object is explicitly constructed, its attribute-value pairs (called "properties") are explicitly given, along with how the value is asserted. For example, the line labeled (1) states that `Idea` is a subclass of itself. (`assertZ(val)` means "insert `val` as the new *last* value in the list of values for attribute `isA`". There is also `assertA(val)`: "insert as the new *first* value", and `assert(val)`: "erase all previous values and only hold `val`").

Lines (2a) and (2b) show inherited values: `subAssertZ(val)` specifying not values of `Idea` directly, but values of its instances.

```
final While
{*
  isA->assertZ(PredefCmdLoop);

  ideasImplicitConstructorA->
    subAssert
    (^ImplicitConstructor
      [*
        [[commandsTestA, Idea],
         [commandsBodyA, Idea]
        ]
      *]
    );
*};
```

Figure 2. Implicit construction.

Figure 2 illustrates SP8b's implicit construction of the value inherited by instances of `While` for attribute `ideasImplicitConstructorA`, in this case `While`'s own constructor. Implicit construction creates anonymous instances of classes where the constructor call fills in values for attributes listed in the constructor. Implicit construction is used extensively to define stylized knowledge efficiently. It is also used to define procedural knowledge like flow control and functions.

Implicit constructor calls begin with a caret (`^`), have the class name, and then the arguments wrapped between `[*` and `*]`. This particular constructor call initializes attribute `implicitConstructorsAttrUTypeListA` (not shown, but given in `ImplicitConstructor`'s definition) to the list of lists:

```
[[commandsTestA,Idea],
 [commandsBodyA,Idea]]
```

What is being constructed is `While`'s own constructor. The first argument to a `While` constructor call will be its value for attribute `commandsTestA`, and the second argument, its value for attribute `commandsBodyA`. Thus

```
^While[* true,stdOut
 ->printLn("Hi!")*];
```

is an infinite loop that prints "`Hi!`".

All objects may be elaborated upon further with subsequent explicit constructions until the keyword `final` is given in the last one (as shown for `While` in figure 2). `final` means no more assertions may be made for that object. For classes, there is also the keyword `noMore`, meaning no more instances of the class may be created.

From the start SP8b was designed to (1) support communities of scientists, (2) handle scientific knowledge like distributions, iterative simulations, meta-data and meta-knowledge, and (3) treat explanations as first-class scientific objects. Support for communities of scientists comes from SP8b's huge shared virtual address space and its internal grammatical knowledge structures which attempt to make it language independent.

SP8b programs are intended to reside on a server. Running an SP8b program follows the client-server computing model. This allows one user's SP8b knowledge structures to be selectively "published" on the server for other users to access. Users may define groups of other users, assign the work done during a particular session to one of those groups, and specify user/group/other permissions in the style of the Unix file-system.

SP8b programs are composed of objects stored in a virtual 320 bit "address space"[3]. We intend to allow *all*

2 A web interface for this system is under active development.

3 It is different from an address space in that SP8b "addresses" are variably-sized. The addresses of sparsely-described identities will be smaller than those of richly-described ones.

interesting[4] work of *all* users to be saved forever. (If 320 bits are not enough, then we have built-in upwards compatibility to increase it.)

Starting a new session causes a 256 bit hash function[5] to be computed from the user name, log-in time, a counter[6], and other parameters to uniquely identify that new session. Objects created during that session share the same 64 bit "page" and are generated at sequential virtual addresses from 0 to 2^{64}-1.

Having a scientific community's work on a single virtual server facilitates collaboration. The full arc of documentation and publication – from lab notebook to textbook – is gathered in one virtual spot, in one form. Anything is viewable by anyone (permissions permitting), with considerably less effort than following a tree of references in printed journals, or even conventional online links.

Further, the SP8b attempts to be natural language independent. The 320 bit address uniquely identifies each frame, and is called its *identity*. At each address could lie either an *idea node* (a data-structure that is a node in a frame system) or a node of a container data-structure (*e.g.* a linked list or binary tree node). Separate dictionaries map identities to natural language terms; thus the system can support multiple languages.

As much as possible, we have avoided making SP8b output hard-coded strings of text in English (or any other natural language) in favor of building structures that may be translated by grammars into strings for natural languages. A primitive grammar for English currently exists, and we intend to follow it up with grammars for Spanish, German and Japanese.

The second major design principle of the SP8b is greater support for scientific knowledge like distributions, iterative simulations, meta-data and meta-knowledge. Why do scientists turn to computers? (1) for their speed, (2) for their precision, and (3) for their ability to handle large datasets. Our 320 bit "address space" was developed with large datasets in mind. As for speed, a dedicated, single-purpose program will always have faster *run-time* speed than SP8b. However, if you want to minimize the developer's (knowledge-engineer's or scientist's) time, then perhaps SP8b is a better option. Additionally, SP8b is designed to facilitate scientific knowledge representation and reasoning by supporting iterative simulations and distributions, checking and computing meta-data and meta-knowledge, and doing *thoughtful* garbage collection.

Although its syntax is different, SP8b has been developed with contemporary C/C++/Java/C# programmers in mind. It has C-inspired flow control instructions corresponding to `for` (`For`), `while` (`While`) and `do-while` (`Repeat`) loops, as well as the `if-else` (`If`) conditional. It also has a `Test` statement meant to encode rules or decision trees concisely. `Test` is in some ways more general and in other ways more restrictive than C's `switch` statement.

This contrasts with languages like Scala, where "elegant" programs minimize variable usage and emphasize recursion. Scientific simulation programming relies heavily on variables and iteration.

Another basic design principle of SP8b is to minimize the number of basic data-types by favoring more general ones over less general ones. Thus:

- Instead of an integer data-type, there is a rational type. Integers are just rationals with denominators of 1.
- Instead of a set data-type, there is a bag type, which is a cross between sets and lists. Bags are like sets in that items are unordered. Bags are like lists in that the same item may be present more than once. (Bags may, of course, be used as if they were sets.)
- Instead of a simple floating point type, there will be a distribution of floating points. Binary operations (other than equivalence) on distributions grow the resulting distributions combinatorially until some maximum distribution size is reached. After that, results are stochastically sampled.
- Instead of arrays, there will be maps. If the map's domain has a fixed finite size, then amortized access time is guaranteed to be O(1) (physical memory limitations permitting).

No floating point (distribution) equivalence operator exists. Instead, equivalence between floating point values will be checked algebraically, by a theorem prover, based on the distribution's derivation, and on domain knowledge.

Although rationals, floating point distributions and non-numeric "symbols" are all idea nodes, a distinction is made between unannotated and annotated values. *Unannotated values* are the rationals, floating point distributions and identities themselves. For example, 2.0 is an SP8b unannotated value. *Annotated values* are unannotated values which have been augmented by meta-data, like dimensionality, units, limits, attributes, *etc*. For example, `2.0(*meters)` is an SP8b annotated value. Operations on values automatically checks meta-data for compatibility (*e.g.* attempting to add 2 meters to 2 seconds is illegal), and generates values with the meta-data appropriately computed.

Slightly different operators exist for unannotated and annotated values. For example, ordinary addition is defined over unannotated numbers. However, for annotated values representing numbers, this is insufficiently specific. Instead, four types of addition are defined:

- Group addition: creating a collection of things from two existing collections,
- Delta group addition: increasing the size of a collection

4 A subject term that is more precisely defined below.

5 Currently SHA-256.

6 In case of collision: when the hash function regenerates a number already in use, the counter is incremented and another hash value is computed.

of things by adding a semantically (if not numerically) small collection to it,
- Extension addition: creating not a collection, but a new single thing, whose attribute is extent of the first and second combined, and,
- Delta extension addition: extending the degree of some thing by a semantically small amount.

While all four of these addition operators add 2+2 to 4, they differ in how they manipulate their corresponding meta-data. For example, the algorithm for group addition does a set union among the sets of two lists of sets. For more details, please see Phillips 2010 and Phillips 2011.

Naturally, identities differ in their importance. Some are intermediary results along the way to some uninteresting computation, like temporarily used iterators. These are *uninteresting*, and unworthy of saving. Others result from a user calculation on some empirical data or object. These are all *potentially interesting*.

Conventional garbage collecting languages use algorithms that recycle memory which is no longer being pointed to from the outside. This approach is too crude for SP8b for several reasons, including the fact that all objects have pointers to them from the outside since everything is in the ontology. Instead, an object's position in the ontology dictates its default interestingness, which in turn automatically determines whether it may be garbage collected.[7]

SP8b will get double-use from the infrastructure for garbage collection, even for objects not subject to automatic garbage collection. Counters in objects that keep track of how many times other objects refer to them may be used to compute an overall information content of a knowledge base. This can be used as a gross Occam's Razor heuristic: a combination of terms and prediction corrections, used to describe a scientific knowledge base, which the system can try to minimize.

SP8b computations will automatically generate a trace of their reasoning. This trace can both be displayed to the user for them to double-check, and can be run as an answer-generating program for stochastic computations.

5. Experiments

SP8b's large virtual address space, and its ability to selectively publish results, are straightforward programming problems inspired by decades-old file-system permission schemes. Either they work properly or do not, and thus will not be examined here.

Because both the SOM8b and SP8b are under development, presently insufficient knowledge bases and insufficient reasoning abilities are coded to properly test justification graphs.

This leaves the testing of its meta-data handling. Since Phillips 2010, there was an open question regarding how much meta-data handling slowed computation. Here, we report on an effort to find out.

In designing the infrastructure for manipulating meta-data, we face a common conundrum of quasi-compiled system builders. The more we hard-code in our implementation language (currently C++), the more speed we gain, yet the more flexibility we lose, in defining how SP8b handles meta-data.

For these experiments, we chose to use libraries written in SP8b. This approach accomplishes two things: (1) it establishes a base-level of performance: inflexibility of implementing meta-data handling in C++, in the future, should be outweighed by speeds significantly faster than this, and (2) it shows users how to write their own meta-data libraries.

All experiments were done on a Lenovo R61 laptop with 3GB RAM and an Intel dual-core T7100 1.80GHz processor running Fedora 16 Linux. The program was compiled with GNU gcc 4.6.3 with optimization level O2.

Data on 694 exoplanets (planets not orbiting our Sun) was download on or shortly after 2013 January 30th from http://exoplanets.org/table. The planetary minimum mass was extracted along with its units (fraction of the size of Jupiter) and references (URLs). This data was translated into SP8b as files, with all of the references as independent objects, and information on 7, 70 or all 694 planets.

```
^Do
[*[
  ^VarDecl[* @iter,  Iterator *],
  ^VarDecl[* @sum,   Value    *],
  ^VarDecl[* @count, Rational *],
  ^VarDecl[* @value, Idea     *],
  @count := 0,
  @sum := 0.0,
  ^For
  [*
    @iter :=
      Planet->
        localProp_iter(hasInstance),

    !@iter->iter_isAtEnd(),

    ^Do
    [*[
      @value :=
        @iter->iter_value()->
                 localGet(minMassA),
      ^If
      [*
        @value->isInstanceOf(Value),
        ^Do
        [*[
          @sum :=
            @sum +grp @value,
          @count := @count +ext 1
        ]*]
```

7 Garbage collection is implemented but primitive.

```
        *]
      ]*],

      @iter->iter_advance()

    *],

    stdOut->printLn(""),
    stdOut->print("Sum:   ")->
              printLn(@sum),
    stdOut->print("Count: ")->
              printLn(@count)
  ]*];
  quit;
```

Figure 3. Summation loop.

We ran a loop that computed the sum of the minimum mass. Not all of the planets had their mass listed, so the loop checked for that (see figure 3). For the "*No meta-data*" case, the loop was run with unannotated values of `minMassA`: no meta-data was computed. This measured the loop's time, apart from meta-data computation.

For the "*Units, subject and attribute*" case, the loop was run on annotated values which computed the units and attribute by checking that they matched. It also computed the subject by looking for a class that subsumes the subjects of both operands (see Phillips 2010).

For the "*All meta-data*" case, the loop was run on annotated values with the units, subject and attribute computed as above, as well as references. Reference symbols were kept in a bag that was used as a set (single occurrences only). Bag sizes grew monotonically.

All cases described above were run 9 times. The first run was always thrown out (to discount any filesystem buffering issues). The table below shows the averages and standard deviations of the times taken by the latter 8 runs. (The summing loop was timed, file parsing was not.) All values are times in seconds.

	n=7	**n=70**	**n=694**
No meta-data	0.1102 ±0.00024	0.5459 ± 0.00077	5.003 ±0.048
Units, subject and attribute	0.2815 ±0.0016	3.126 ±0.016	60.90±8.7
All meta-data	0.4656 ±0.0020	4.706 ±0.035	235.5±68.

Table 1.

We interpret these results as follows:

(1) As expected, computing meta-data makes calculating more expensive. Meta-data computing loops took 3-47 times longer. Assuming the worst figures are due to virtual memory, we are heartened that the marginal cost of additional meta-data seems to be relatively low. Although computing the resulting set of references is O(n lg n), for the n=7 and 70 cases "*All meta-data*" only ran 65% longer than the "*Units, subject and attribute*".

(2) For the n=694 cases of "*All meta-data*" and "*Units, subject and attribute*" we believe the system took significantly longer because of virtual memory. The system has a primitive garbage collector. We expect these times to go down as it is made more sophisticated.

One interpretation of this is that meta-data computation is too expensive: if not in time, then in memory. That interpretation, however, assumes that computing resources are more valuable than a scientist's time. Had we summed numbers with nothing in common – times, distances, and masses of planets, insects, and atoms – SP8b would have alerted us of trouble. This ability becomes more important as computation becomes more deeply embedded in large, inhomogeneous, computational frameworks. SP8b can check (and convert) units, and detect errors in assumptions.

Still, the problem of the poor times remain. The garbage collection algorithm is primitive. We believe that as it is refined and database access are implemented, this problem will no longer manifest.

Alternatively, perhaps we can gain both the speed of unannotated computation with the correctness of annotated computation, by defining functions that operate over whole classes, not just two operands at a time. We expect data of instances from the same class to have similar meta-data. This could be recorded at the level of the class, or checked by algorithms that were optimized with this assumption. Then, sums or other computations could be done with the efficiency of unannotated values, but with the security of knowing meta-data was handled correctly.

6. Discussion

Here we discuss design decisions, as well as the methodological one: "Why a frame-based system instead of a Bayesian approach?" One potential criticism is that the semantic web obviates the need for an "operating system" for scientific reasoning like SOM8b: the role of SP8b could be done by an XML-based markup language. This, however, is a misunderstanding of the scope of our research. We seek not only to specify meta-data (*e.g.* units) but also knowledge, like algorithms. Thus, on one hand, our scope is extremely broad: a marriage of XML-like mark up with something like the Java Virtual Machine's algorithm generality. On the other hand, we are consciously limiting ourselves to science.

Another potential criticism is that because we intend to incorporate a theorem prover, why not build our system around one? The answer is that deductive reasoning is only a subset of scientific reasoning. We believe that a meta-reasoner that calls deductive reasoners, as needed, better models how scientists actually behave.

A third potential criticism is on the use of a fixed-length address space. We did this purely for simplicity. And, as all knowledge currently resides on the same virtual server, there is less of a need to point to resources outside of itself. (It can refer to resources outside with URL strings. However, when that data or knowledge is internalized, it is assigned its own page. Humans find specifying a 320 bit number difficult, so the system uses "Local Resource Locators" like natural language strings.)

The Bayesian approach is perhaps the most well known alternative to the frame system approach to scientific knowledge representation. The Bayesian networks or "graphical models" have been widely adopted, especially for diagnosis applications, such as medicine and for mechanical failures (Szolovits 1978, Isermann and Ballé 1997). Such approaches, however, generally assume a very specific type of wisdom, and a very specific type of ignorance. The wisdom is that the principles of the system under study (*e.g*. an automobile, a human body) are well understood, and that sufficiently similar populations have been studied from which relevant statistics have been gathered. The ignorance is that while we can readily observe symptoms of a malfunctioning system, we cannot observe its exact state because it is either too costly and time-consuming to dismantle (*e.g*. the car), or because doing so would cause more harm than good (*e.g*. vivisection on the human). When used diagnostically, Bayesian reasoning's power derives from applying knowledge of both how a system grossly works (the graph's layout), and about the population (the priors), to make an educated guess about a data-poor individual.

We acknowledge Bayesian reasoning's power, yet we seek not to be constrained by its limitations for adequate domain knowledge, to build a graph and adequate distribution knowledge, and to populate it with priors. We believe the best solution is to give the system the ability to build its own Bayesian networks as needed.

Fortunately, SP8b supports the traditional programming flow-control structures and variables to implement Bayesian network building algorithms.

We believe that default reasoning is a better model of how scientists grapple with their grander issues: *"What is light exactly: particle, wave, or kind of both?", "Are the number of species fixed or can they evolve over time?", "Does phlogiston have negative weight or should I believe in Lavoisier's 'oxygen'?"* We believe that it is an abuse of Bayesian network notation to require *a priori* enumerations of all alternatives, and it is impossible to assign prior probabilities, in a principled fashion, for such cases. Default reasoning, however, lets us proceed. "*Let us assume option 1, and go as far as we can. Now let us try something else.*"

Kuhn tells us that most scientists most of the time are doing "normal science": extending the current paradigm instead of overthrowing it. And, in such cases, we intend to use the *logic* portion of default logic as far as it can go, and even to be able to use Bayesian reasoning and related tools, such as stochastic simulations, to extend our reasoning further.

7. Conclusion

Although SP8b is unfinished, it shows early promise. It successfully computes meta-data, and can compute additional meta-data attributes for a marginal additional cost. We intend to implement measures to increase the scale of computations which our system can compute, with features like thoughtful garbage collection, database access, and improved operations.

Further, we look forward to handling justifications as we would other scientific objects, and to scaling the system up to handle multiple users.

References

Frenkel, Karen A. 1991. The Human Genome Project and Informatics *Communications of the ACM.* 34(11), Nov. 1991. pg 40-51.

Friedland, Noah; et al. 2004. Project Halo: Towards a Digital Aristotle *AI Magazine (AIM)*, 25(4): 29-48.

Gunning, David; et al. 2010. Project Halo Update – Toward Digital Aristotle *AI Magazine*, 31(3): 33-58.

Giere, Ronald. 1999. Visual Models. *Science without Laws*. Chicago, IL, USA: University of Chicago Press. Chapter 7.

Isermann, R., Ballé, P. 1997. Trends in the Application of Model-based Fault Detection and Diagnosis of Technical Processes. *Control Engineering Practice*. 5(5). 709–719.

Korpela, E.; Werthimer, D.; Anderson, D.; Cobb, J.; Lebofsky, M. 2001. SETI@home-massively Distributed Computing for SETI, *Computing in Science & Engineering*. 3(1). Jan/Feb 2001. 78 – 83.

Kuhn, Thomas. 1962. *The Structure of Scientific Revolutions, 2nd Edition*. Chicago, IL, USA: University of Chicago Press. Chapters 3 and 4.

Mayr, Ernst. 1988. The Multiple Meanings of Teleological. *Toward a New Philosophy of Biology*. Cambridge, MA, USA: Harvard Univ. Press. Chapter 3.

Phillips, Joseph. 2010. A Proposed Semantics for the Sampled Values and Metadata of Scientific Values. *Midwest Artificial Intelligence and Cognitive Science Conference*. 16-23.

Phillips, Joseph. 2011. Towards a Technique of Incorporating Domain Knowledge for Unit Conversion in Scientific Reasoning Systems. *Midwest Artificial Intelligence and Cognitive Science Conference*. 154-159.

Szolovits, Peter. 1978. Categorical and Probabilistic Reasoning in Medical Diagnosis. *Artificial Intelligence*. 11(1-2). 115–144.

van Fraassen, Bas C. 1980. The Pragmatic Theory of Explanation. *The Scientific Image*. New York City, NY, USA: Oxford University Press. Chapter 5.