Propagation: A Revolutionary Model for Computation

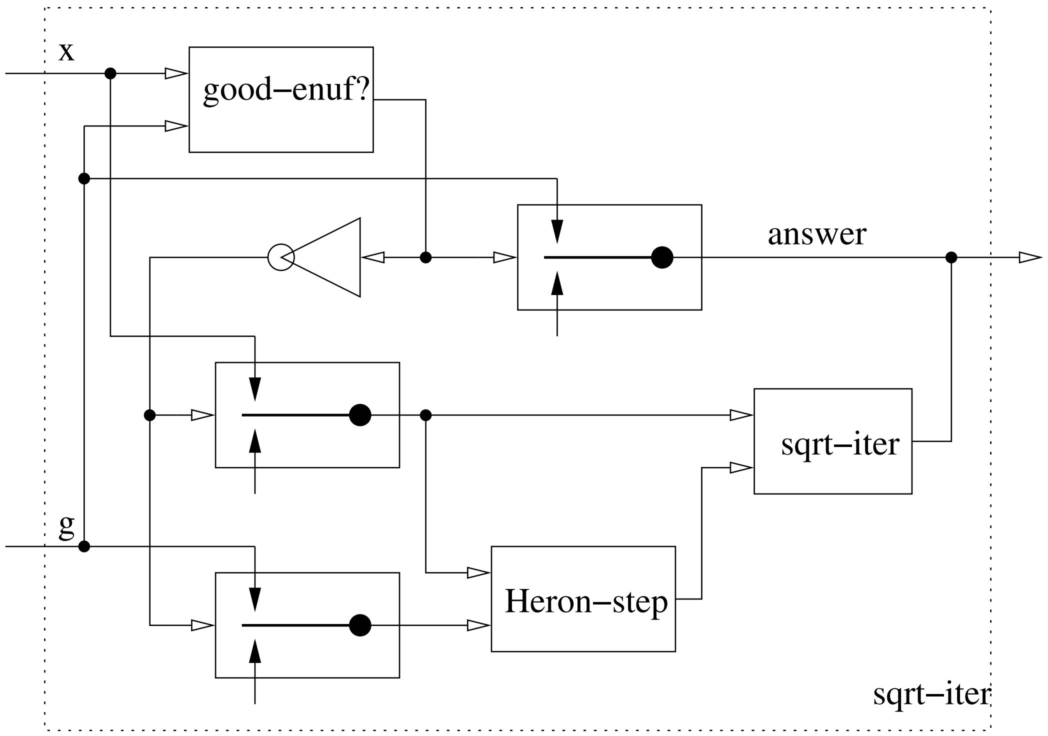
Seedling Proposal
Gerald Jay Sussman, MIT CSAIL & EECS

Vision

Instructors in introductory programming subjects often draw attention to similarities between programming and recipe writing. Too bad. It helps perpetuate a way of thinking in place since Grace Hopper invented COBOL. True, we have introduced abstraction, the model-view-controller idea, and integrated development environments, but those are all incremental, not revolutionary.

Absent a revolution, we continue write programs the way we always have, and those programs are narrow and brittle. Other sorts of engineered and natural systems, in contrast, are often robust and adaptable. The Internet, for example, has adapted from a small system to one of global scale. Our cities evolve organically, to accommodate new business models, life styles, and means of transportation and communication. Indeed, from observation of biological systems we see that it is possible to build systems that can adapt to changes in the environment, both individually and as an evolutionary ensemble.

Why not shift the paradigm so that we can design and build computational systems with ideas ready to be borrowed from engineering practice? In particular, we believe the time has come to focus on *propagation*, a computational model built on the engineering idea that the basic computational elements are autonomous machines interconnected by shared cells through which they communicate, as in the following:



Each machine in a propagation system continuously examines the cells it is interested in, and adds information to some based on deductions it can make from information from the others. This model makes it easy to smoothly combine expression-oriented and constraint-based programming.

Traditional computational models allow the value stored in a "place" to come from only one source. By "place" we mean the variables, the components of compound data structures, and the implicit transient storage given to the intermediate results of subexpressions. A "source" is whatever corresponding construct produces the value that is to be put in the place. In the presence of mutation "place" must be interpreted in spacetime: the value in a place during any particular period when that place is not mutated comes from just one source, and when the place is mutated, the new value also comes from just one source.

If we allow places to receive values from multiple sources, an individual source need no longer be responsible for computing the complete value that goes into a place. Indeed, some partial information about a value can already be useful to some client, and can perhaps be augmented by some other source later (which perhaps used that partial knowledge to deduce the refinement!). Also, if places can accept values from multiple sources, we need not decide in advance which computation will end up producing the value that goes into a place. We can instead construct systems whose information flow depends on how they are used.

The idea that a place holds partial information, rather than either nothing or the completed result of a computation is a fundamental change to the infrastructure of computation. A system built on propagators can accommodate such paradigms as functional computing, constraint propagation, and SAT solving. It can be a flexible and expressive substrate for systems that maintain provenance of data, and provide advanced support for debugging, security, auditing, and accountability. It can accommodate optimal implicit search to support exploratory behavior.

Anticipated Contributions and Representative Impact

Our goal beyond the seedling is to produce a naturally concurrent and distributed model and infrastructure for computation that will make it easier to build systems that are reliable in the face of natural failure and deliberate attack. We will have support for auditable and accountable systems that are robust and adaptable to novel applications. We will have a computing paradigm that supports novel exciting applications.

Our general claim is that propagators offer a paradigm shift in the way programming is done, and inasmuch as computing is everywhere, everything will be touched by a revolution in computing. The following suggest particularly important areas ripe for early application:

Web-enabled information superiority with propagators maintaining truth. In information-rich, web-exploiting systems, conclusions are rarely final: new information and new sites are always entering the system, rendering any sort of closed-world hypothesis inoperable. Moreover, the massive scale and distributed architecture of the web ensure that reasoning engines will encounter contradictions, so that provenance and truth maintenance must be addressed as a matter of course. The propagator framework subsumes these mechanisms as part of the base language implementation, which frees programmers from the need to manage them explicitly.

Evolution of complex systems with propagators promoting incremental replacement. Military systems are big and monolithic. Consequently, they are mightily costly to replace, but often impossibly difficult to evolve, with the net result that they are often obsolete in total just because they are obsolete in part. The propagator box-and-wire paradigm enforces a new kind of modularity that enables new software to coexist with old, either by box replacement or box collaboration.

Increased security with propagators exposing flows. No single idea will solve the security problem, but the wires in propagator-based systems provide natural points of interest for monitoring systems aimed at noting suspicious patterns of behavior.

Steps toward a propagation-based, engineering-inspired model

In the 1990s we developed Amorphous Computing, where large numbers of asynchronous processing elements are locally interconnected in irregular ways. This gave us some deep understanding of how distributed algorithms may be built that are independent of the local connectedness of a system. During the past two years, Beal and Sussman have demonstrated the use of controlled-hallucination wrappers for generalizing system components. And during the past few months, Radul and Sussman have developed and published papers about a new propagator model that seems to accommodate all other known programming models in a natural way.

Our next steps are clear. Our current propagation models are data-push models: computation occurs if it can. But to be efficient we also want need-pull models: computation is only done if the results are needed for some other reason. We will develop a request-acknowledge protocol that allows us to combine these. We are encouraged because requests are themselves a kind of data that can propagated, with dependencies, but there are real problems lurking here that may be difficult to resolve.

Our current propagation models are simulated with sequential processes. We will build and study truly concurrent versions to understand the engineering tradeoffs implied by communication costs. We must demonstrate that our propagation model is scalable to large systems with multiple processors and distributed memory.

Our current propagation models are expressed in a monolithic linguistic structure. We must evaluate our claim that we can accommodate and combine alternative programming models so that they can cooperate in a unified system.

Also, we must understand the problems of resource allocation and deallocation implied by propagation models. For example, how should we arrange garbage collection of propagators and cells in a distributed, shared, multiuser environment?

Seedling demonstration

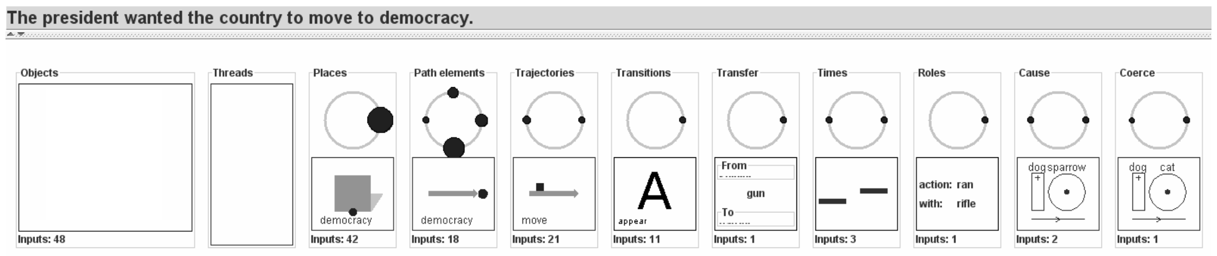
Several criteria for a seedling-level demonstration come to mind:

- Demonstration involves one or more systems of larger than illustration size, to demonstrate scalability.

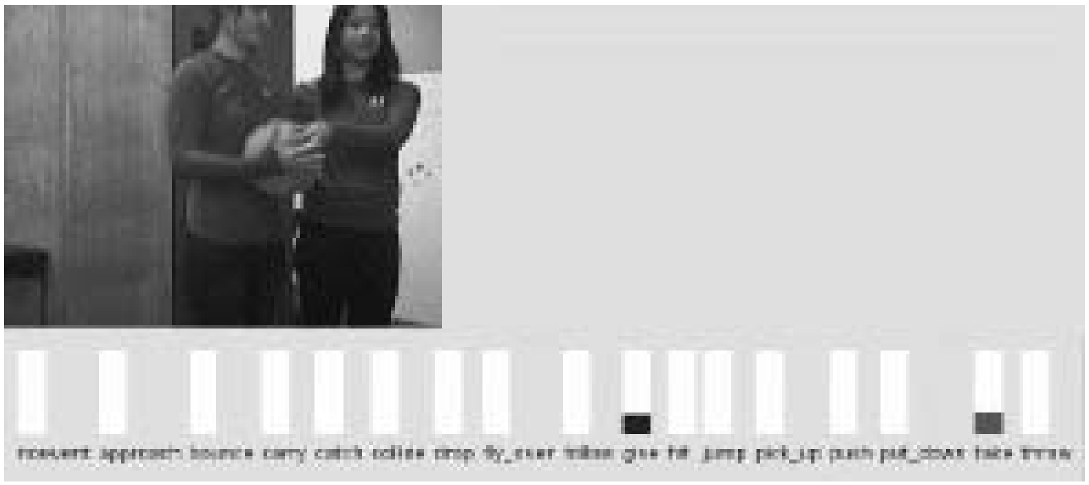
- Demonstration involves a system of systems, to demonstrate value in large-scale development.
- Demonstration involves replacement of one large subsystem with another, to demonstrate incremental replacement.
- Demonstration involves interaction of systems via the web, to demonstrate value in assembling web-based components.
- Demonstration involves systems written in diverse languages, running on multiple machines to demonstrate flexibility.
- Demonstration involves systems of interest to DARPA.

Accordingly, we propose a concentrated effort to rework a medium-sized concept-oriented research system and to tie that system to large, well-established language parser on one end and on the other end to a relatively small, rapidly evolving, vision and spatial reasoning system.

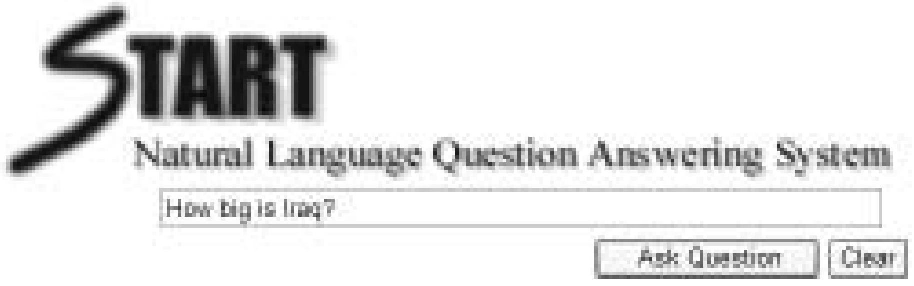
The Genesis language and concept-formation system learns from examples how to translate the results produced by syntactic parses into a dozen semantically explicit descriptions of class, change, place, path, trajectory, transfer, time, cause, coercion, containment, mood, and role. The system, developed by Patrick Winston and his students in Java, is a prime candidate for the center of the seedling generation because its architecture is already informed by the propagator idea; because it contains on the order of 75 distinct components; because a central goal is to form a tightly coupled loop that grounds understanding in vision and spatial reasoning, because it uses an off-the-shelf, unexpectedly unreliable syntactic parser that we are extremely eager to replace, and because the time has come to connect it with a vision and spatial reasoning program now emerging from another DARPA seedling. In the illustration, the system interprets a sentence involving an abstract trajectory.


The president wanted the country to move to democracy.

The Genesis vision and spatial reasoning system, developed by Sajit Rao and his students in C++, learns to recognize instances of actions such as give, take, jump, push, and drop involving two students and a ball. The two sides of the Genesis system—language and vision—are meant to demonstrate the power of grounding language in vision and visual imagination and the power of conditioning vision with problems and expectations delivered via language. In the illustration, the system judges a video to be, with equal likelihood, a *give* and a *take*.


noevent approach bounce carry catch collide drop fly_over follow give hit jump pick_up push put_down take throw

The START system is a publicly-accessible information access system that has been available for use on the Internet since 1993 (http://start.mit.edu/). The system, developed by Boris Katz and his students in Lisp, answers natural language questions by presenting components of text and multimedia information drawn from a set of information resources that are hosted locally or accessed remotely through the Internet. These resources contain structured, semi-structured and unstructured information. Start itself is a prime target for downstream exploitation of the propagator idea, as natural language naturally involves ambiguities that have to be resolved by multiple experts. For our purposes, however, START is to be viewed as ported box, accessed over the web, which receives sentences on a wire and puts one or more syntactic analyses on another wire. In the illustration, the system prepares to answer a question about Iraq.



We satisfies all the listed criteria by connecting the Genesis language system, the Genesis vision system, and the START system and by making explicit use of propagator idea in the Genesis language and concept formation system.

Work Statement

During the 12-month seedling effort, Professor Gerald J. Sussman will lead the conceptual development of the propagator idea along with Alexy Radul (post doc as of June, 2009) and three students.

They will:

- Report on development of basic data-pull implementation.
- Report on designs for implementations with true concurrency.
- Deliver propagator 1.0 Scheme code for use by others.

Professor Patrick H. Winston will lead the Genesis–Language—Genesis–Vision—START demonstration working with three students.

They will:

- Rebuild the language-and-concept portion of the Genesis system on an explit propagator foundation.
- Instrument the explicit propagator foundation so as to observe and study behavior patterns.
- Connect the Genesis language-and-concept system to the START web server using a propagator riding on an `http` substrate.
- Connect the Genesis language-and-concept system to the Genesis vision-and-spatial reasoning system using a propagator riding on a remote-procedure-call substrate such as SOAP.
- Report on gains experience by using propagator ideas.
- Deliver propagator 1.0 Java code for use by others.