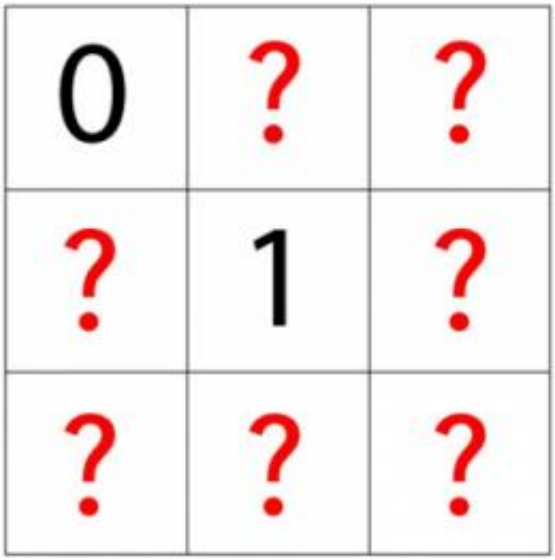
Computing, Sudoku-style

April 28 2010, by Larry Hardesty



When Alexey Radul began graduate work at MIT's Computer Science and Artificial Intelligence Lab in 2003, he was interested in natural-language processing -- designing software that could understand ordinary written English. But he was so dissatisfied with the computer systems that natural-language researchers had to work with that, in his dissertation, he ended up investigating a new conceptual framework for computing. The work, which Radul is now pursuing as a postdoc in the lab of Gerald Sussman, the Matsushita Professor of Electrical Engineering, is still in its infancy. But it could someday have consequences for artificial-intelligence research, parallel computing and the design of computer hardware.

Artificial-intelligence systems, Radul explains, often tackle problems in stages. A natural-language program trying to make sense of a page of written text, for instance, first determines where words and sentences begin and end; then it identifies each word's probable part of speech; then it diagrams the grammatical structure of the sentences. Only then does it move on to stages with names like "scope resolution" and "anaphora." The process might have a dozen stages in all.

In a multistage process, however, errors compound from stage to stage. "Even if they're really good stages, they're 95 percent," Radul says. "Ninety-five percent is considered extraordinary." If each stage is 95 percent accurate, a five-stage process is 77 percent accurate; a 20-stage process — by no means unheard-of in AI research — is only 36 percent accurate.

Systems that can feed information from later stages back to earlier stages can correct compounding errors, but they're enormously complicated, and building them from scratch is prohibitively time consuming for most researchers. A few such single-purpose systems have been designed for particular applications, but they can't easily be adapted to new problems.

Branching out

Radul envisioned a new type of computer system that would handle multidirectional information flow automatically. Indeed, not only would it pass information forward and backward through stages of a multistage process, but it would pass data laterally, too: The results of one stage could be fed into, say, two others, which would attack a problem from different directions simultaneously, reconciling their answers before passing them on to the next stage. At that point, the stages of a process wouldn't really be stages at all, but computational modules that could be arranged in parallel or in series, like elements in an electrical circuit.

Programmers would simply specify how each module was connected to those around it, and the system would automatically pass information around until it found solutions that satisfied the constraints imposed by all the modules.

This reconception of programming, however, required a commensurate reconception of computation. Classically, a computer is thought of as having two main parts: a logic circuit and a memory. The logic circuit fetches data from memory, performs an operation on the data, and ships back the results. Then it moves on to the next chunk of data. In Radul's system, on the other hand, multiple logic circuits and memory cells are arranged in a large network. Any given logic circuit can exchange data with several different memory cells, and any given memory cell can exchange data with several different logic circuits.

The danger with this arrangement is that logic circuits storing data in the same memory cell may arrive at contradictory conclusions. Which conclusion should the memory cell store? Instead of working together to solve a problem, the logic circuits could end up simply overwriting each other's data.

In the prototype system that he developed for his doctoral dissertation, Radul solved this problem by devising memory cells that don't so much store data as gradually accumulate information *about* data. One logic circuit, for instance, might conclude that the value of some variable is between five and 15; the memory cell will register that the number it's storing falls within that range. Another logic circuit, with access to the same memory cell, might conclude that the value of the variable is between 10 and 20; the memory cell would thus contract the range of the value it's storing to between 10 and 15. A good analogy might be someone solving a Sudoku puzzle, who's identified two or three candidate values for a puzzle square and jots them in the corner, expecting to winnow them down as new information comes to light.

Owning up

A programmer using Radul's system is free to decide what kinds of data about data the memory cells will store. But in his prototype, Radul enabled the memory cells to track where data comes from, a capacity that he thinks could be useful in a wide range of applications. In explaining this aspect of the system, Radul assigns the logic circuits arbitrary names. Say that a group of three logic circuits — Alice, Bob and Carol — converged on a value between 10 and 15 for some variable, but a fourth circuit — Dave — assigned the variable a value of 237. The system could warn the entire network that the results of Dave's calculations are suspect and should be given less weight until new information propagating through the network brings them in line with everyone else's. (It's also possible, however, that the new information could vindicate Dave and force Alice, Bob and Carol to revise their initial conclusions.)

Again, the Sudoku analogy might help. Sudoku solvers sometimes make mistakes; but once they've identified a mistake, it may already have propagated across the whole puzzle. Radul's system would, in effect, automatically back out all the other errors that flow from the original mistake.

Radul's network of logic circuits and memory cells is an abstraction: It describes how information flows through a computer system, not necessarily the design of the system's hardware. It so happens, however, that computer chip manufacturers have reached the point where the only way to improve performance is to add more "cores" — or logic circuits — to each chip. Splitting up programming tasks so that they can run, in parallel, on separate cores, is a problem that has bedeviled computer scientists. But a mature version of Radul's framework would allow programmers to specify computational problems in a way that automatically takes advantage of parallelism.

“All of computing — all of it, object-oriented, parallel, all those kinds of computing,” says Daniel Friedman, a professor of computer science at Indiana University, “they put all the responsibility on the programmer.” With a system like Radul’s, however, “large hunks of the responsibility would likely go away.” Friedman cautions that “there’s a huge amount of research to be done to demonstrate all that. All they’ve done so far is demonstrate it on a very small scale.” But, he adds, “this is spectacular stuff. I’m just looking for the right student to come along to get all fired up about it.”

Provided by Massachusetts Institute of Technology