become active at any level—or even at several levels. For example, consider a virus written in a high-level computer language. When a program containing the virus is compiled, it results in a machine-language program that searches for other programs in the high-level language and inserts the high-level version of itself into them. Such tampering in human-readable files is very easy to detect, but it offers the virus one great advantage: the ability to spread from one kind of computer and operating system to an entirely different one, since other compilers can translate it into other machine languages.

There are more subtle ways to achieve machine independence. A cleverly doctored compiler can insert the viral code into the programs it prepares. Modern compilers are often written in their own language and compiled by older versions of themselves. An archetype is the C compiler that constructs the popular Unix operating system. Partly because it is so easy to install on new machines, Unix has grown to be the most widespread operating system. Unix and C were invented in the early 1970s at Bell Labs by two young hackers, Ken Thompson and Dennis Ritchie, on their own initiative. AT&T, the owner of Bell Labs, took very little notice of it at first, until the system, improved by many programmers, had spread to thousands of machines worldwide. Finally, in the late 1970s, AT&T decided to sell Unix systems commercially. As part of the formalization of the product, AT&T assigned a programming team to extend and optimize it. During the course of careful scrutiny, someone noticed that the C compiler produced a large, completely unexplained block of machine-language program when it reached a certain point in compiling another C compiler. The discovery led to a number of closed-door meetings. In 1984 Ken Thompson, in a lecture on the occasion of his receiving the Turing Award (sort of a computer-science Oscar) shed light on the issue.

The programming team had uncovered a fiendishly clever virus, one designed to allow Ken Thompson to bypass security and to log into any Unix system running on any machine, including future machines yet to be invented. The mysterious block of code that appeared in new C compilers had two purposes. One was to cause a copy of itself to appear in any future C compilers that it itself compiled. The other was to insert a block of code that would respond to a password known to Ken Thompson, at a certain place in any log-in program that it compiled. The beauty of the scheme was that these

blocks of code were reproduced not directly in machine language, which would work on only one type of machine, but by means of an ephemeral C version of themselves that was immediately translated by the compiler into the machine language of whatever machine it was compiling for. This self-reproducing C program had once existed in an actual computer file in Ken Thompson's machine but now had only a ghostly existence, reappearing momentarily deep in the innards of the computer whenever a C compiler executed the viral code and immediately vanishing again.

A ghost virus haunting a compiler is very hard to find even if one knows what to look for because a compiler is a huge program, and the virus appears only as an innocent-looking piece of code somewhere in the middle of a vast tract of legitimate machine language. The principle could be used to construct viruses with other purposes that could spread undetected between different kinds of machines. This example also hints at the possibility that even more subtle and elaborate creations are possible. Computers can bring mathematical abstractions to vigorous life, and there are no mathematical limits to the subtlety and deviousness. The fun has just begun.

## Spontaneous Generation

So far I have confined this review of software parasites to those that are deliberately constructed, because these constitute the majority of the wildlife known to date. But such fabrications are limited by the imaginations of their human creators. Our increasingly complex systems are capable of creating their own surprises, and in times to come we can expect shockingly original gremlins to arise spontaneously in our intelligent machinery, the result of unexpected interactions or mutations of existing parts. A few stirrings have already been observed.

The ARPAnet, funded by the Department of Defense's Advanced Research Projects Agency, is a computer network created in the late 1960s to allow computers at facilities across the country to communicate with one another. The goal is to share scarce resources. Computers at individual sites are connected to the ARPAnet through special small computers once called Interface Message Processors, or IMPs. The IMP of one site is connected (through leased lines) only to those of a few nearby sites. Fancy software allows messages to cover greater

distances by being handed rapidly from IMP to IMP. There are many possible indirect routes from point A to point B in the net. Depending on fluctuating traffic, sometimes one route is faster, sometimes another. To help make instantaneous routing decisions, each IMP maintains a table that records how long it takes recent messages to travel from the IMP through alternative routes to other sites. The table is updated with information obtained partially from the tables of neighboring IMPs. The network is monitored and maintained under contract by a company in Cambridge, Massachusetts. Operators at this site can stop, examine the contents, reload, and generally fiddle with any IMP on the net through special priority messages routed via the net itself. This generally works well, and even the most serious problems (such as large power outages affecting several IMPs) are handled smoothly.

In 1972 (and again in 1980, and probably other times as well) a plague hit the ARPAnet. The symptoms were that net traffic became hopelessly congested around a site in the Los Angeles area (in the 1980 incident the locus was Boston). Network control, suspecting some glitch in the program of the machine at the focus of the congestion, shut it down, reloaded its program, verified that it was working correctly, and reconnected it to the network. The problem persisted. Indeed, it seemed to be spreading outward from the original IMP. Shutting down and reloading larger numbers of IMPs did not fix things either; the congestion continued its spread and returned to the original sites as soon as they were reactivated. The network seemed to be haunted by a very persistent ghost. Many unsuccessful experiments later, order was finally restored by shutting down the entire network, clearing all the memories of all the IMPs, reloading their programs, and starting fresh—like sterilizing a whole planet with death rays, then seeding it with new life!

A subsequent analysis revealed what had happened. The focal IMP in Los Angeles had a memory error and developed an erroneous entry in its routing table. The table now indicated that messages sent via this IMP would experience a large *negative* delay. Next, adjacent IMPs calculated that it was more advantageous to send messages via this IMP than directly, since its negative delay more than made up for the extra hops. IMPs connected to *those* then decided it would be best to transmit via Los Angeles, and so on. The error in the initial IMP rapidly spread to the routing tables across the land. Wiping out the memories of a few IMPs did not clear the problem, because

the erroneous numbers would spread back from IMPs that were still affected. Or infected. In fact, the network was inhabited by a spontaneously evolved, quite abstract, self-reproducing organism. This organism was formed by a simple, random mutation of a normal, sanctioned piece of data. It did not even involve a programming language.

The plague was easily spotted and eradicated because its effects were so devastating. If it had been more subtle in its action, it might have lived much longer. Among programs without masters there is a strong natural-selection criterion: reproduce but lie low. It is quite likely that many unsuspected organisms are already making a quiet living among the abstraction hierarchies in computer memories everywhere. Most will never be discovered. The plague also suggests a quick way in which a wild information organism can come into being, namely, by a mutation in an existing self-replicator. Since any datum in a computer is subject to duplication, this covers a lot of ground. A human-created computer virus, existing in many copies spread widely through different computer systems, would be a particularly fertile candidate for a liberating mutation. If the part of its code that caused it to make trouble were inactivated by a mutation, it would be less likely to draw attention to itself and thus would be more likely to reproduce indefinitely. The mutation might even make it unrecognizable, and thus safe, from a virus killer. Further mutations that eliminated unnecessary code, and thus reduced its size, would improve its chances even more. In time it might change in such a way as to fine-tune the frequency and kind of mutations it undergoes, making it more adaptable. Run-ins with other programs could endow it with major new pieces of code, and new capabilities. In time it might even acquire the ability to systematically copy and try out fragments of code from other programs and other viruses—the beginnings of computer-virus sex!

Such examples merely demonstrate the limits of our imagination. The most effective organisms would be much more subtly encoded and would escape detection entirely. From time to time one might expect one to surface because it developed a nasty side effect. That kind of mutation would generally prove fatal for the organism. As intelligence in programs progresses, we should also expect program fragments that can plan and act in a deliberate, calculating, and creative manner to enhance their survival. The data realm will host