This book began with a discussion of the “swarm” that confronts us in networked life. The faceless foes that inundate networks make it difficult to imagine how one can adequately face up to and deal with the other. One of the premises of *Ethical Programs* is that the predicament of hospitality is the primary ethical problem that addresses networked rhetorical situations. That predicament presents itself over and over again, in the form of the Law of hospitality, and we continually attempt to answer it by way of the laws of hospitality, the contingent responses we author in order to deal with the swarm. The arrival of others scrambles what we might typically conceive of as a rhetorical situation—a clearly understood context in which a rhetor addresses an audience toward some particular end. The problem of defining context is not new, it is not specific to the study of rhetoric, and it is not created by the messy spaces of networked technology. Nonetheless, the hospitable network does raise a number of questions about how one understands and theorizes a rhetorical situation.

For instance, during a February 2007 press conference, President George W. Bush discussed the complexities of knowing one’s audience. Bush was answering questions about his proposal for a “troop surge” that would increase the number of U.S. troops in Iraq. Recent debate about the surge had led White House press secretary Tony Snow to ask whether congressional remarks about capping troop levels would make it easier for Iranians or al Qaeda members to make inroads in Iraq. Snow had been criticized for trying to silence debate, since his remarks suggested that any opposition to Bush’s plan would embolden the enemy. When asked about Snow’s remarks, the president said the following:

The only thing I can tell you is that when I speak, I’m very conscience [sic] about the audiences that are listening to my words. The first audience, ob-
viously, is the American people. The second audience would be the troops and their families. That’s why I appreciate the question about whether or not—about the troop morale, it gave me a chance to talk to the families and how proud we are of them. Third, no question people are watching what happens here in America. The enemy listens to what’s happening, the Iraqi people listen to the words, the Iranians.

In a remark that does not directly address whether Snow’s remarks were intended to have a chilling effect on the debate, the president gestured toward the complexities of understanding one’s audience in a world shaped by information networks. This same set of problems arose during Mitt Romney’s infamous remarks at a fundraiser about the 47 percent of Americans that would never support him because they represented a segment of the population “dependent upon government, who believe that they are victims, who believe the government has a responsibility to care for them.” When video of these remarks leaked, Romney quickly discovered that the hospitable network means that one’s remarks can be distributed in unknown and complex ways. While Bush seemed to be aware of the multiple audiences he was always addressing, Romney appeared to have (at least temporarily) forgotten this fact. Regardless, politics in the hospitable network means that the rhetor has already welcomed multiple audiences to the rhetorical situation, audiences that exceed the intended audience and that arrive with conflicting and competing interests.

These problems of audience are directly tied to the predicament of hospitality, to the problems that emerge when we recognize that networked life is founded on the Law of hospitality. But though Bush seems to have at least a somewhat nuanced understanding of the networked rhetorical situation (even if he is using such an understanding to raise questions about the danger of arguments opposing his policies), he still conceives of that situation as a collection of discrete entities. This approach is understandable as we attempt to address the complexities of contemporary information environments. Networked life invites others, meaning that audiences, texts, and contexts become infinitely more complicated. However, this understanding takes what Jenny Edbauer calls a “conglomeration” approach to understanding rhetorical situations by simply adding more and more entities to such situations. For Edbauer, this approach ignores the ways that components overlap and intersect and how they “bleed into one another.” To address this situation, she suggests that we theorize not rhetorical situations but rather rhetorical ecologies in the interest of “add[ing] the dimensions of history and movement (back) into our visions/versions of rhetoric’s public situations, reclaiming rhetoric from artificially elementary frameworks.” Rather than adding elements to
our models of rhetorical situations, Edbauer argues for understanding persuasion and communication in terms of overlapping and intersecting ecologies, allowing us to “more fully theorize rhetoric as a public(s) creation.” If rhetorical action helps to create and maintain publics, then Edbauer insists that we take up a more complex, ecological approach to understanding the processes by which this happens.

Ethical Programs has taken on the complexities of rhetorical ecologies by focusing on how computational artifacts help to shape and constrain rhetorical action in the network. My focus on software is not meant to suggest that software creates such ecologies or that it is purely determinative of rhetorical action. Instead, the previous chapters have shown that software bleeds out into other dimensions of a rhetorical ecology, affecting how rhetors read, write, argue, or persuade. It is nearly impossible to separate software out from our rhetorical ecologies or to consider it separately from other important overlapping components of our rhetorical ecologies. For instance, chapter 2 demonstrated how the Obama campaign’s software used procedural rhetoric to funnel volunteers to certain kinds of activities. These procedural arguments were not confined to the software itself but were also locatable in the phone-banking scripts distributed to volunteers. By following the procedural rhetorics of the campaign through software and beyond, tracking them even to the procedures authored by volunteers, that chapter demonstrated how procedures bled out and transformed, entering into overlapping and intersecting rhetorical ecologies. This example shows that any decision to focus only on software would limit our ability to understand how protocological power circulated through the campaign.

Software serves as the starting point for each of the rhetorical analyses presented in this book, and this is in part an answer to Ian Bogost’s critique of work in digital rhetoric that has failed to address computation. I take up this critique in more detail in the next section, but for now I want to note how Ethical Programs continues the approach of much work in software studies, focusing on situated activities in order to ask broader questions about computation and software. The authors of 10 PRINT CHR$(205.5+RND(1)); : GOTO 10, a text that uses a single line of code in order to explore the history of creative computing, explain that this method is the opposite of much work in the digital humanities. While a great deal of digital humanities scholarship uses computation to address large sets of texts and other media, 10 PRINT and other work in software studies instead operate “as if under a centrifugal force, spiraling outward from a single line of text to explore seemingly disparate aspects of culture.” This focus on the specific serves to guard against bombastic arguments, which often tell us less about how particular technologies op-
erate than they do about the theoretical and political agendas of the arguers. Rather than making large claims about software’s role in networked life, the approach enacted in this book insists on sitting with specific rhetorical ecologies and tracing their effects.

Of course, the danger of this type of approach is that the results might not be generalizable and that we learn something about a particular situation or piece of software without learning something broader about computational life. One answer to this critique lies in the use of the term “centrifugal” by the authors of 10 PRINT, which describes how work in software studies can take something as minute as a single line of code as a launching point for discussions about disparate cultural trajectories. However, another way to address this problem of the particular and the generalizable is to embrace the specificity of such approaches and to call for many more of them. Each attempt to describe and analyze specific ecologies involving software may provide only a sliver of understanding, a snapshot of how computational artifacts operate as cultural artifacts. However, in aggregate such studies help us build an archive that can aid scholars across disciplines develop methods and approaches for studying software as a cultural form. Given the relative youth of software studies, we are only at the early stages of developing such an archive, but the willingness of scholars in the field to embrace multiple approaches is already bearing fruit.

If a focused analysis of particular ecologies is one of the central methods of software studies, then rhetorical studies has much to offer such conversations. Software and rhetoric both benefit from an ecological approach in two ways. First, both rhetoric and software help to shape, enable, and constrain what is possible in a given rhetorical ecology. Rhetoric is the study of the available (or, as I argue in chapter 3, possible) means of persuasion, the study of what we can do or say in a given space, but it also offers strategies for transforming that possibility space and inventing new arguments and approaches. Similarly, software lays out a possibility space, whether it’s a videogame that uses procedures to control certain behaviors or a word processing program that determines what is available to a writer. In the case of both software and rhetoric, an ecology of action is shaped and, in some sense, coded (and recoded). However, both software and rhetorical action are “ecological” in another sense: they are crafted as responses to overlapping (and sometimes conflicting) exigencies. It is now a commonplace in software studies and in new media studies more broadly that computer programming is much more than a specialized technical practice used to create tools. That is, software is not just the background for rhetorical action, the thing we use to get things done. As an authored artifact, software is also the result of rhetorical action; it
is the medium for expressing ideas and making arguments. Software is thus woven through rhetorical action. It is the result of rhetorical action, since it uses computation as an expressive medium. But, as I have suggested above, software also helps to launch and distribute what we normally think of as rhetorical action: the distribution of arguments across media channels. Thus, software sits chiasmatically between different types of rhetorical action, simultaneously the result of and grounding for our attempts at persuasion and communication. Given that rhetoric and software are tied together in the creation, maintenance, and (sometimes) disruption of our attempts to interact, it becomes clear that rhetoricians and software studies scholars have much on which to collaborate.

Still, the focus on particular ecologies is not a way of ignoring “the swarm” of hospitable networks by focusing on a single entity or object. Carried out in a narrow way, such analysis could be seen as grounding rhetorical ecologies in software. But grounding a rhetorical ecology transforms it back into a situation, conceiving of it as something easily bounded and understood. As the previous chapters have demonstrated, software is not determinative of rhetoric and ethics in networked life. Software establishes ethical programs, but such programs are often (though, not always) manipulated and reconfigured by actors in a rhetorical ecology. Beginning from software is not an attempt to simplify complex rhetorical action but is instead an attempt to open up new paths of inquiry, demonstrating that computational artifacts can be the starting point for understanding complex ecologies, even as that analysis sometimes spins out and gives way to centrifugal force. Beginning rhetorical analysis from computational artifacts focuses our attention on an entity that is often seen as part of the background, something that scholarship in software studies has successfully called into question.

What the analyses in this book have offered are demonstrations of how the language and methods of rhetorical studies provide a particularly useful way of drawing our attention to computational artifacts. Software addresses (and helps us to address) the complications of hospitality, which stand as the primary ethical predicament of networked life. The ethical programs enacted by such software—programs that I have described as attempts to author the laws of hospitality—reconfigure our rhetorical ecologies, calling for new persuasive strategies and shaping what can or cannot happen in a given space. However, to this point, the subtitle of this book—“Hospitality and the Rhetorics of Software”—has remained in the background. While I have argued throughout for different understandings of how software intersects with rhetorical theory and practice, I have not yet explicitly defined the rhetorics of software that emerge in networked life. In the remainder of this concluding chapter, I
remedy this situation, moving from the local approach of earlier chapters to a more global approach that presents some terms and concepts for tracking the rhetorics of software.

The Rhetorics of Software: Procedural Rhetoric and Beyond

In addition to providing a way for rhetoricians to understand the persuasive dimensions of computation, Bogost’s *Persuasive Games* also offers a powerful critique of much of the work in digital rhetoric. Citing a number of works in the field, Bogost argues that much of this scholarship ends up “mistaking subordinate properties of the computer for primary ones.” That is, digital rhetoricians have tended to focus on the use of computers to write, create images, or distribute arguments rather than on how computation itself can be used to craft arguments:

> In short, digital rhetoric tends to focus on the presentation of traditional materials—especially text and images—without accounting for the computational underpinnings of that presentation. . . . digital rhetoric must address the role of procedurality, the unique representational property of the computer.

For Bogost, a key example of this is Richard Lanham’s *Electronic Word*, one of the first texts to explicitly link rhetorical theory to digital technologies and a text that opened the way for a generation of digital rhetoricians. Lanham focuses on how computers can manipulate the appearance of text (something Lanham himself insists is not unique to digital technology) and does not attend to what new rhetorical possibilities emerge in computational environments. When Lanham demonstrates how the manipulations of text on screen allows us toggle between looking “at” text (noticing surface and style) and “through” text (reading for meaning), he is not necessarily telling us much about the rhetorical affordances of the computer itself. Instead, he is linking computational technologies to the rhetorical tradition without considering how such technologies might actually introduce novel rhetorical theories of persuasion, communication, or identification.

Bogost’s argument has been a necessary corrective for the field of digital rhetoric, and I myself have found the concept of procedural rhetoric extremely useful in understanding both the rhetorical capacities of software and the rhetorical nature of procedures more generally. As I have argued, procedural rhetoric becomes a useful rhetorical strategy for navigating the complex and contradictory power relations of protocological networks. However, my analysis in this book has aimed to examine the rhetorics (plural) of software.
exploits, which expose what’s possible in a given space, to ethos, which presents the digital rhetor with resources for living and arguing in deep archives, to machinic thinking, which allows for movement between the worldviews of database and narrative, I have argued for a more expansive understanding of the rhetorics of software. Procedural rhetoric offers one inroad for rhetoricians analyzing software, but it is only one of the possible rhetorics of software. In the interest of making explicit this notion of the rhetorics of software, I offer here a discussion of these different rhetorics. Given that the hospitality of networked life provokes, shapes, enables, and constrains rhetorical action, I describe these rhetorics in terms of the Law of hospitality and the laws of hospitality. If the hospitable network scrambles our rhetorical situations and forces us to instead attend to rhetorical ecologies, and if software helps to enact the ethical programs that engage the hospitality of networked life, then we require new ways of understanding how software links up with different levels of rhetorical action.

To this end, I offer three rhetorics of software—arguing about software, arguing with software, and arguing in software—as a way of productively re-expanding digital rhetoric beyond procedural rhetoric. These three realms of rhetorical action are not discrete or separable; all the intersecting rhetorics of software participate in complex rhetorical ecologies.

Arguing about Software

This is the rhetorical realm that most would imagine as the rhetorician’s primary purview: how we talk about software. While such work offers only one way of understanding the broader cultural implications of software, examining how we talk and argue about software is important. Given that more people are programming (beyond specialized disciplines such as engineering and computer science), these conversations will no doubt become more interdisciplinary and will invite the expert and the novice. The Law of hospitality extends a broad invitation. Who will accept this invitation and what laws will be authored in response? Here, hospitality is playing out in a particularly striking way. Projects such as Code Academy, which found New York City mayor Michael Bloomberg among its pupils in 2012, are now inviting many to learn how to write code. This means that conversations about software design are expanding outward, calling on both experts and novices to analyze, critique, and write software. In addition to this mixing of expert with newcomer, our hospitable networks are also inviting interdisciplinary conversations about code, and software studies presents a key set of methods for enabling these conversations.

However, for arguments about software to happen in a productive way,
all parties will require a deeper understanding of computational processes and functions. Recent work in new media studies has pushed beyond the interface, attempting to correct decades of work that treated the screen as another page. This effort is ongoing, and there is more to be done. However, these same efforts can and should be extended to spaces and conversations outside of the academy as well. This means studying (and perhaps even intervening in) arguments about software, examining how such conversations are conducted, noticing where they succeed and fail, and accounting for how power dynamics shape these arguments. The questions one might ask when approaching arguments about software are the following: How do conversations about software happen? Who participates, and who doesn’t? How inclusive are these conversations? What power relations shape arguments about software? What commonplaces circulate? Who are the overlapping and intersecting audiences for such arguments?

In previous chapters, we have seen a number of arguments about software. Chapter 3 offers some of the clearest examples of how these conversations happen in networked spaces and how such conversations will always have to respond to the predicament of hospitality. The Twitter onMouseover exploit invited a far-reaching conversation about software, one that drew in journalists, software designers, and everyday users of Twitter. One discursive space—the website Stackoverflow.com—is of particular interest when considering arguments about software. A site that combines the functions of a wiki and a message board, Stackoverflow.com offers novice and expert a space to discuss software, to ask questions, and to enter what we might view as a complex set of master-apprentice relationships. This space may or may not cultivate a nurturing space for budding programmers or curious tinkerers, and that is largely linked to the hospitality described in this book. Networked life means that we bump up against others, whether or not we want to. The results of these collisions may bring happy results or they may alienate newcomers, but a space like Stackoverflow.com at least allows for the possibility that arguments and discussions about software can happen among those from different backgrounds. When the onMouseover exploit occupied Twitter users for part of a day (and when I myself began studying the exploit and its aftermath), this site became a useful resource for those seeking explanations. These arguments and discussions about the exploit were one level of that particular rhetorical ecology, one of the rhetorics of software circulating around this event.

Arguments and discussions about the other exploit examined in chapter 3—the OAuth exploit—took a decidedly different shape. While the onMouseover exploit forced a conversation that mixed expert with novice and programmer with user, the OAuth exploit happened among a relatively small circle of programmers, designers, and executives. While this exploit triggered arguments about software, those arguments circulated quite differently, and
the result of this discussion was that the exploit was addressed prior to wreaking havoc on users. Happening behind closed doors before being shared with a broader public, this argument about software revealed a very different rhetorical ecology and involved different kinds of power dynamics, but the results of the OAuth exploit were no less important to users than the results of the onMouseover exploit. In fact, we could argue that the implications of the OAuth exploit were much more far reaching, even if a smaller number of people were involved in how it was addressed. The OAuth exploit had the potential to expose a great number of users to a security vulnerability that would have allowed a third-party application to act on their behalf. While the onMouseover exploit demonstrated a relatively low stakes vulnerability, the hack of OAuth could have caused much more harm. Understanding how conversations emerge and happen in the wake of such exploits is crucial for the digital rhetorician interested in understanding how software is discussed, addressed, and reshaped by rhetorical exchange.

While chapter 3’s discussion of exploits offers one of the clearest examples of arguing about software, we saw this same level of rhetorical activity at play in discussions of the MyBO software, as critics analyzed how the campaign used different systems to motivate volunteers and to guide volunteers to certain activities. These conversations happened among people involved in political campaigning, but they also happened among software developers whose primary interests and expertise lie outside the realm of campaigning and volunteer coordination. Once again, this was a wide-ranging discussion involving multiple constituencies and levels of expertise. Discussions among journalists regarding Narrative Science’s robot journalists fall into this category as well. Those discussions reflected a great deal of suspicion for many reasons, not the least of which was that journalists felt that their livelihood was being threatened by algorithmic journalists. This was a completely understandable (and in many ways, justified) response, but it is a response that would benefit from a deeper and broader discussion of code and computation. Narrative Science itself is an interesting model for this kind of conversation, given that it employs both programmers and journalists. The company’s robot writers are the result of many conversations about software, conversations that inform how the software is written. This movement from arguing about software to the practice of composing the software itself brings us to our next level of rhetorical activity.

Arguing with Software

Arguing about software is never sealed off from how we argue with software, how we use software, code, and computational procedures to make arguments. If arguing about software becomes more complex because of a hos-
pitable network that both welcomes and shuns participants to a discussion, then arguing with software increases in complexity because of the hospitable network’s penchant for simultaneously welcoming and turning away hacks, exploits, and other bits of code. Hospitable networks mean that hacking and exploring the possibilities of computational environments are within the scope of rhetorical action. But the “with” here can also be taken another way. While we can use software as a tool for expression and argumentation (as in when we say we are writing “with” a pencil), we might also see software systems as an interlocutor (as in when we say we are arguing “with” another person). These two uses of “with” are not separate—when I attempt to use computation to build an argument, I work with and struggle against the affordances of a given platform or language. While the end result might be an argument expressed by way of software, I have a number of rhetorical exchanges with the software itself—as I write code, I’m writing for multiple human and machine audiences. Arguing with software is a concept meant to account for this entire complex set of relations.

To again take the Twitter exploits of chapter 3 as an example, both conversations about these exploits and the exploits themselves are part of the various, circulating, and overlapping rhetorics of software in this ecology. When Kinugawa and others created exploits that exposed a flaw in Twitter’s URL parser, they were using software in the interest of rhetorical action. They were both attempting to persuade Twitter to address a security flaw and helping to reveal the possible means of persuasion in this space. Invited by networked software that continuously negotiates between absolute and measured hospitality, between the Law and the laws, these ethical programs expose what is or is not possible in a given space. But more than just initiating a discussion about software (arguing about), Kinugawa, Holm, and the others who circulated the onMouseover exploit used computation itself to make arguments, and they did so by negotiating with the software itself. They performed an ethical argument by way of computation (software as tool) by interacting with and exploiting gaps in the software itself (software as interlocutor).

The exploit is one particularly powerful way of arguing with software, but it is of course coupled with procedural rhetoric. Here the Obama campaign’s use of social networking software to funnel users to particular activities comes to mind. The MyBO software was certainly a tool for more efficiently organizing volunteers, but it was also a collection of procedural arguments made by the campaign about how (or whether) volunteers might engage opposing arguments. Given a networked campaign infrastructure that purported to invite volunteers to help craft and shape a message, this use of procedural rhetoric is all the more interesting. For while the campaign argued with software, using procedural rhetoric to help control a protocological network, volunteers also
used procedural arguments to write back against the campaign and navigate the complex power dynamics of networked power. The volunteer-authored procedural arguments covered in chapter 2 are not examples of arguing with software (even if they are examples of procedural arguments), but this only further demonstrates that all of these rhetorics of software intersect and overlap with one another. Engaging the procedural arguments expressed, in part, by software, Obama campaign volunteers used a similar strategy by translating those strategies into other media, such as phone-banking scripts.

MediaWiki also offers an example of arguing with software, though in this case the arguments are less overt and not as loaded with a particular political program. By considering each keystroke as crucial to the integrity of the database and by welcoming all of this data into its deep textual archive, MediaWiki engages the predicament of hospitality by attempting to absorb all information and not determining ahead of time what is or is not useful or relevant. When usernames and article titles are considered equal in the eyes of the database, we can begin to see MediaWiki as a piece of software that responds to the Law of hospitality by crafting a set of laws, laws that use software to make arguments about how textual discussions should happen and what information should be tracked. The designers of MediaWiki may or may not have had these particular arguments in mind when building the software, but the arguments are nonetheless there, shaping what can or cannot happen as writers edit and create articles or conduct conversations about policy. Chapter 4 explains how this plays out in the most famous MediaWiki installation in the world—Wikipedia. If MediaWiki is an argument made with software, that argument reveals itself in complex ways as Wikipedians try to argue in this software environment (a level of rhetorical activity taken up in the next section). MediaWiki’s hospitable database also means that users must engage in arguments with the software itself, and we see this most clearly as Larry Sanger’s Citizendium project attempts to work both with and against the affordances of the software. Having to build a credentialing system on top of a system that is designed to exclude credentials as an elevated form of evidence means that the software becomes a party to rhetorical exchange. Sanger and the designers of Citizendium’s MediaWiki installation had to argue and struggle with MediaWiki in order to build the kind of dwelling they envisioned.

The questions a rhetorician might ask when exploring arguing with software are myriad: How hackable are computational spaces? What kinds of ethical assumptions and arguments are made by software platforms? Who or what is able to manipulate and exploit such platforms? Who or what stands at the thresholds, determining what can or cannot happen in networked spaces? What are the means of persuasion available to those coding and recoding computational environments? How does software design change if we begin
Arguing in Software

Software helps to shape, enable, and constrain rhetorical action. Given the ubiquity of software, an increasing number of rhetorical ecologies are happening in software environments, meaning that rhetoricians and scholars of all stripes should be attempting to understand how persuasive acts emerge in such environments. The Law of hospitality plays a key role here by welcoming data, by storing and tracking nearly any keystroke. What rhetorical strategies are most effective in such spaces? How do humans and computational programs deal with this Law of hospitality? Software addresses this Law with some of its own laws, by building deep archives of information. But these ethical programs in turn shape those of humans, who must now address the complex rhetorical ecologies shaped by these archives. Networked spaces encourage certain kinds of strategies and modes of argument, and I have traced a number of these throughout this book. Each of these rhetorical strategies emerge out of software environments and the computational spaces that accumulate data. As we interact with software and as that software serves to help shape rhetorical interactions, it becomes important to understand what strategies are encouraged and foreclosed by software.

In MediaWiki’s deep archive, ethos emerges as a crucial rhetorical strategy. The controversy surrounding Essjay shows us that MediaWiki’s penchant for textual accumulation makes ethos the primary way by which Wikipedians both make and critique arguments. MediaWiki is driven by an ethic of database integrity, and it operates most effectively when all information is archived. Even malicious edits are often retained in the edit history of Wikipedia articles, demonstrating that even seemingly “useless” data is kept and archived. In another example of this somewhat radical hospitality, those Wikipedians who attempt to change usernames and identities are thwarted by an archive that keeps track of username changes. Logs of user activity link old usernames to new ones. MediaWiki’s ethical programs are best understood in terms of this guiding ethic—the desire to archive rather than to delete. There are cer-
tainly exceptions to this rule, but this generalized ethic of preservation stands as what best defines this software dwelling (its ethos). Essjay was arguing in this environment, and he benefited from this archive as he built up credibility as a tenured academic. Anyone curious as to why he was carrying weight in a discussion need only have looked at his user page or his previous edits to learn that he was a theological expert. However, he was undone by this same aspect of Wikipedia’s dwelling, and his ethos crumbled when it was discovered that his situated ethos—the portion of ethos that is supposed to precede the rhetor and to be outside of his or her control—was in fact an invented ethos. MediaWiki played an important role in both the creation and destruction of Essjay, and that role is best understood in terms of how Wikipedians argue in software, taking advantage of (or suffering at the hands of) the particular affordances of software environments.

The examples from chapter 4 make arguing in software even more complex, since the robot journalists circulating through networked life are operating alongside humans in rhetorical ecologies. The algorithms authored to generate news stories are an example of arguing with software—of crafting computational procedures that make arguments about which information is most important and how a narrative should be structured. However, I have argued that the bots themselves present us with key strategies for operating, persuading, and communicating in software environments. By using procedures to sift and sort data and by transforming data into narratives, these bots remind us that the generation of narrative and argument is always, in some sense, machinic. Rhetoric, as a set of procedures for generating and critiquing arguments, offers a long tradition of understanding how to move between the worldviews of narrative and database (although ancient rhetoricians would of course never have put it in these terms). While arguing in software environments that welcome data and that intensifies a situation of “information overload,” users can turn to the “meta-writers” of Narrative Science and to the increasing number of computational machines generating narratives for inspiration. Given that the relationship between narrative and database is increasingly complex, we can turn to machinic thinking as we develop strategies for arguing in software.

The case of procedural rhetoric is also interesting in this regard, since it becomes both a way to argue with and in software. We can craft computational processes that make arguments, and I have demonstrated the complexity of such actions in the preceding chapters, showing that exploits, procedural arguments, and less explicit arguments in platforms like MediaWiki demonstrate how computational procedures make arguments and engage the Law of hospitality. However, the phone-banking scripts of Obama campaign volunteers show how we can also craft other types of processes that make argu-
ments. The authors of these scripts engaged (or chose not to engage) opposing arguments, and these choices were made by volunteers who accepted the campaign’s invitation to participate. Stepping into a software environment (MyBO) that generated lists of possible voters, clicking through those lists, and making phone calls, the actions of these volunteers were shaped in important ways by the campaign’s procedural rhetorics. However, the phone-banking scripts were very different kinds of procedures in that volunteers did not execute them in the same way a computational machine executes a procedure. Instead, volunteers rewrote the scripts in the interest of different kinds of procedural arguments. Here procedural rhetoric sits both in the computational environment of the campaign’s software and in the volunteers’ use of procedures to engage possible voters. Such hinge points between the different rhetorics of software—the different layers of rhetorical activity in networked, computational spaces—present us with rich possibilities for cultivating complex accounts of rhetorical ecologies. Understanding procedurality as a mode of inscription that can be enacted in both computational media and discursively is but one place where we might notice software’s promise as a participant in rhetorical ecologies.

Given how much contemporary rhetorical action happens in software environments, the rhetorician of software can begin to ask: What are the affordances of certain software platforms, and how do users take advantage of or fall prey to those affordances? What persuasive strategies are opened up and closed off by certain software environments? How do user activities lead to changes in software platforms, shifting the available means of persuasion? What does the wide-scale adoption of certain software platforms (from Microsoft Office to Facebook) mean for the possibilities of rhetorical action? How do users take up software in unexpected ways, revealing expressive possibilities within an environment in ways that designers never imagined?

The Future of Ethical Programs

My hope is that these intersecting and overlapping rhetorics of software offer a way forward for those of us interested in examining the ethical programs of networked life. Digital rhetoricians can and should be participating in discussions of computation, and they should do so both by bringing rhetorical theory to bear on software and by rethinking rhetorical theory in light of the unique attributes of computational media. However, while my account of the rhetorics of software offers some possible futures for the analysis of networked software, there remains a different (and perhaps more difficult) question about the future of ethical programs.

What is the future of any ethical program? I don’t ask this question in or-
der to make predictions or proclamations about how software will be used or rewritten or recoded in the face of the Law of hospitality. I’m asking a different question, one that considers whether the use of software to institute ethical programs opens up the possibility of a future, of an “unprogrammed” encounter with others. We noted briefly, in chapter 1, that there is a general distrust of computation in many discussions of ethics. Computation is seen as something mechanistic and perhaps even inhuman. A computer can enact procedures, can make rules and follow them, and any responsible ethics would have to move beyond such a stable, inflexible program. Given this, can an ethical program have a future? Or does a program decide beforehand what will happen, closing off the possibility of reimagining what is or is not possible? If software enacts a set of rules that defines a possibility space, that determines how others will be dealt with, then does it allow for novel approaches to continuously arriving ethical questions?

While Levinas set the human against the computer, suggesting that enacting a set of rules too cleanly answers the infinite question of ethics because it is “something of which a computer is capable,” then Derrida offers a somewhat more hospitable consideration of computational media. This book owes everything to Derrida’s work on hospitality in texts such as Of Hospitality, On Cosmopolitanism and Forgiveness, and Adieu to Emmanuel Levinas. However, it is another text, one that takes up the question of “the animal,” that is most applicable to the future of ethical programs. In The Animal That Therefore I Am, Derrida is concerned with the limits drawn between “so-called human” and “the animal.” His analysis of Kant, Heidegger, Levinas, and Jacques Lacan suggests that each of these thinkers can be traced back to a common “father,” namely René Descartes. Each of these thinkers, attempting to undo the Cartesian project, finds himself a Cartesian when it comes to the nonhuman animal.

However, what is of most interest given my own analysis of software, rhetoric, and ethics is how Derrida invokes computation in his interrogation of the multiple limits drawn between “human” and “the animal.” His treatment of computation allows us to consider how open Derrida’s analysis is to the possibility that a machine may have a future—that a mechanism that is programmed to react to situations by way of code will perhaps open the way toward an “authentic” future that has not already been decided. So, for instance, Derrida examines one of the limits between animal and human by taking up Descartes’s insistence that the animal cannot “respond.” The human can respond to a question while “the animal” can only react, a distinction of which even Descartes seems unsure. Derrida tracks Descartes’s treatment of response and reaction through not only the famous Discourse on Method but also through Descartes’s letters. In one of those letters, Descartes seems to
hesitate, to consider whether in fact “the animal” can respond. But before turning to this letter, Derrida quotes from a moment in the Discourse on Method in which Descartes presents his own two-step Turing test:

If there were machines bearing the image of our bodies, and capable of imitating our actions as far as it is morally possible, there would still remain two most certain tests whereby to know that they were not therefore really men. Of these the first is that they could never use words or other signs arranged in such a manner as is competent to us in order to declare our thoughts to others; for we may easily conceive a machine to be so constructed that it emits vocables, and even that it emits some correspondent to the action upon it of external objects which cause a change in its organs; for example, if touched in a particular place it may demand what we wish to say to it; if in another it may cry out that it is hurt, and such like; but not that it should arrange them variously so as appositely to reply to what is said in its presence, as men of the lowest grade of intellect can do. The second test is, that although such machines might execute many things with equal or perhaps greater perfection than any of us, they would, without doubt, fail in certain others from which it could be discovered that they did not act from knowledge, but solely from the disposition of their organs: for while Reason is an universal instrument that is alike available on every occasion, these organs, on the contrary, need a particular arrangement for each particular action; whence it must be morally impossible that there should exist in any machine a diversity of organs sufficient to enable it to act in all the occurrences of life, in the way in which our reason enables us to act.10

This quote is fascinating for any number of reasons, not least of which is Descartes seeming interest in interface design—he describes machines and how they might respond to certain inputs. Most importantly, given our present discussion, Descartes believes that human reason can reconfigure its “organs” given a particular rhetorical ecology, situating itself in a way that meets an exigence. That is, a human can respond. A machine (or “the animal,” it makes little difference to Descartes) cannot do this—it can only react by way of “the disposition of [its] organs.” It can only execute its code. Upon taking us through Descartes’s discourse on automata, Derrida turns to a letter in which Descartes rethinks this hard-line stance. Here, Descartes reconsiders his argument that the animal-machine can’t respond, arguing instead that the animal-machine, while it might be able to respond to commands, cannot respond “to questions, questioning concerning ‘what is asked of them.’”11 Thus, the animal might be able to respond to its name, but neither animal
nor machine could “produce different arrangements of words so as to give an appropriately meaningful answer to whatever is said in its presence, as the dullest of men can do.”

This brief detour through Descartes (and Derrida’s reading of Descartes) gets us to the moments when Derrida allows for the future of the computer. First, Derrida notes that Descartes probably “could not have imagined in their refinements, capacity, and complexity all the powers of reaction-response that today we can, and tomorrow we should be better and better able to attribute to machines.” One can only imagine how Descartes would have responded to Apple’s Siri. But more than pointing out technological advances that Descartes could not have foreseen, Derrida signals that machines, like “the animal,” are forcing a wholesale reconsideration of what we think we know about the human. In particular, they are forcing us to rethink what we mean by the supposed human realm of the authentic question—a question that is not programmed or robotic and that is not simply answered by way of a program. What are we asking about when we consider the distinction between “human” responses and “machine” reactions?

The question of the response is thus that of the question, of the response as response to a question that, at one and the same time, would remain unprogrammable and leave to the other alone the freedom to respond, presuming that were possible (a techno-historical field with a bright future, even though the programmation of question and response seems to foreclose the future).

This remark about a “bright” future is, of course, not a celebration. It is instead an acknowledgment of two contradictory ideas: (1) that our only chance at a “bright future” is to pursue the unprogrammable question, the authentic encounter that does not decide the answer ahead of time; (2) that there is no encounter with an other, no ethics, without a program. The Law of hospitality is what allows for the possibility of a future, one that does not decide in advance who can arrive, but the laws authored in response to that Law will unavoidably miss the mark. The laws, as ethical programs, find themselves in this impossible space, attempting to answer an impossible demand.

Derrida’s passage about programming and “bright futures” points to the possibility of a program—ethical or otherwise, computational or otherwise—that does not necessarily decide in advance the entire range of possibilities. This is a program that does not know exactly where it is going. To put it in the terms laid out in this book, it would be an ethical program that is a provisional response to the Law of hospitality. Such a program would, to some extent, define what is possible, and we might find it difficult to locate a radical
“future” in computational machines that execute code. But the Law of hospitality makes it difficult for any ethical program, even one carried out by a computational machine, to remain in place forever. The future of ethical programs requires a continuous vigilance in this regard, a vigilance that insists upon constant reexamination of the laws. For networked life in a computational world, this means attempting to understand how our programs are written, how they lay out what can or cannot happen in a given space. At any given moment, an ethical program might be futureless—it will have already decided what will happen. But the reprogrammability of computational machines allows for the possibility of ethical programs that are open to possibility.

The overlapping and intersecting rhetorics of software traced in this conclusion aim to understand and to write the futures of ethical programs, to understand computation not as a foreclosed space of rules that will always offer a rigid program of action but rather to understand software as enacting contingent responses to the Law of hospitality and to understand our interactions with software as similarly contingent. The laws, even those enacted by computational machines, will always be haunted by the Law. This possibility is our only justification for the hope that any ethical program—computational or otherwise—has a future.