Science and Engineering

in Knowledge Representation and Reasoning

Lynn Andrea Stein
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

The Third International Conference on Knowledge Representation and Reasoning (KR '92) conference was held in Cambridge, Massachusetts, in October of 1992. This article is an edited version of a talk, surveying that conference, which was presented by the author at the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI '93). Although nominally an conference overview, the article attempts to summarize the state of the conference and of the field with respect to the intertwined goals of Science and Engineering. The author points out what she hopes is a growing trend within the KR community to develop science inspired by engineering and to engineer systems according to KR's science.

1 Knowledge Representation: Disconnected Science?

As a field, knowledge representation has often been accused of being off in a theoretical no-man's land, removed from and largely unrelated to the central issues in artificial intelligence. This article argues that recent trends in KR demonstrate instead the benefits of the interplay between science on the one hand and engineering on the other, a lesson from which all of AI could benefit. The article grew out of a survey talk on the KR '92 conference, presented by the author at IJCAI '93.

Nonetheless, this article is not intended exclusively for the KR community. While it contains discussion of some of the technical material presented at KR '92, its focus is on the field as a whole, the direction I see it taking, and the implications of this direction for people both inside and outside of KR. As a survey, this article is both sketchy and heavily biased.[1] It touches all too briefly on only a handful of papers, mostly in support of a central argument. This theme is the relationship between recent work in KR and its supporting pillars: Science and Engineering.

Before going on, I should explain what it is that I intend when I speak of Science and of Engineering. Science, to me, represents the study of phenomena. Science asks what, and why, and how. Sometimes Science is theoretical, speculating on or formalizing hypotheses of what might be; at other times, it is empirical, analyzing what is. But, in and of itself, Science builds only theories. Engineering, its twin and partner, is concerned with the construction of artifacts. Engineering is less about questions than about answers. This is not to say that Engineering is without theory—there are both theories of how to engineer and engineered theories. But Engineering is fundamentally about the result, about what is built, about what is created.

Of course, the problem with this neat dichotomy is that it is not. First, the theories Scientists build to explain what is are themselves artifacts, Engineered. And Engineers can only construct their artifacts—physical or theoretical—by understanding the Science of putting things together. On the whole, Engineering works best when informed by Science, and Science is most useful when advised by Engineering. To sharply separate the two is to miss the whole point.

In KR (and in AI and computer science more generally), there is a second problem with the attempt to distinguish Science from Engineering. Much of KR has traditionally focused on what we might call Science, the investigation of what's there. But—as Simon has so aptly noted—that Science is not a Science of reality, but a Science of the artificial reality that we create. Although AI claims at times to be influenced by Psychology and by Biology—by the reality of biological intelligence—our bias as a field is evident in

[1] Given the number of papers in the conference, the sixty minute talk on which this article is based would have alloted each author significantly less than Andy Warhol's proverbial fifteen minutes of fame — closer to fifteen seconds, in fact!

our chosen name. Thus, our Science is inherently a Science of Engineering, and our Engineering is often difficult to distinguish from our Science.

The KR community, in particular, has over the last decade acquired something of a reputation for obscurity, putatively disinterested in and disengaged from the reality which is supposed to underlie AI. We are dismissed as "logic hackers," presumably in contrast to those "real hackers" who build software artifacts: Scientists when we should be Engineers, theorizing when we ought to build. To build on Simon's observation, KR is accused of going beyond a science of the artificial, of having become artificial Science.

This article is an attempt to convince you otherwise. Whatever the truth of this view in the past, I believe that both those within and those outside of KR need to reassess and reevaluate the direction that the field is now taking. A certain amount of KR is indeed strictly Science with an exclusive focus on the artificial world which we create; this is Science walled off from Engineering. But there is also a great deal of KR which goes beyond this narrow realm; the wall between Science and Engineering has come down. In the remainder of this article, I hope to describe examples of Science and Engineering intertwined. For readers who are AI Engineers, concerned with the practice of AI systems rather than its theory, I hope to demonstrate that the field of KR can usefully inform system building. And for the KR "scientist," I will argue that it is not enough to provide useful input for the engineer, but also that the science of KR itself should be driven by engineering problems.

At the KR conference, and in the field of KR today, Science is informing Engineering, and Engineering is informing Science. And I believe that the KR community is paying a great deal of attention to this cycle. This article is an attempt to encourage that trend within the KR community, and to encourage the AI community outside of KR to pay attention to these developments within KR, to foster them in KR and elsewhere throughout AI. By reopening this dialogue, AI at large will help the KR community, and the KR community in term will help all of AI.

The outline of the rest of this article is as follows. First, I will describe some of the Science that was presented at the KR conference. Then, I will turn to Engineering. The last part of this article will focus on hybrids between Science and Engineering, papers that specifically address ways in which Science and Engineering play back and forth. I will cover very few of the interesting papers that have been published in KR in the recent past; I hope that this article will inspire you to seek those out yourself.

2 Science

There were many papers in the KR proceedings which addressed the fundamental problems of Science: that is, they tried to analyze the way that Artificial Intelligence and cognition—or representation and reasoning—work.

An excellent example of such a paper is "New Results on Local Inference Relations," by Robert Givan and David McAllester (pp. 403–412). This is unquestionably a Science paper. It addresses a phenomenon that exists. Its domain is (PROLOG-like) horn clause programs, and this paper answers many open questions about those programs and about the ways in which such programs may be practical. It builds upon earlier work of McAllester's, which described a very practical but fairly trivial class of horn clause programs: *superficial* rule sets. The KR'92 paper resolves many of the open questions raised by the earlier paper and then goes on to build on those results.

Givan and McAllester define a class of horn clause rule sets that correspond to the class P, the polynomial-time computable functions. These *local* PROLOG programs are in fact equivalent to the class P: every program in P can be characterized by such a local horn clause rule set, and inference over local rule sets is polynomial-time decidable. Givan and McAllester then go on to demonstrate that determining whether or not a rule set is local is an undecidable problem. This is an interesting piece of Science. It tells us a fundamental fact about what there is. Unfortunately, it doesn't really help us if we want to go and build something.

The third result of Givan and McAllester's paper is different. They define a subset of local rule sets, *inductively local* rule sets, which *can* be tractably recognized. And as a consequence, they demonstrate that you can give a rule set to a program which will tell you whether your rule set is inductively local and, if it is, will automatically transform it into a polynomial time decision procedure.

In short, Givan and McAllester's work falls clearly into the Science side of KR. It is, in some sense, "logic hacking;" it's playing with theorems about logic. It does not result directly in an implementation. Nonetheless, this paper has a great deal to say to people who are playing with PROLOG programs. If you are an engineer who uses Horn clauses, this paper is a set of significant results which tells you how you can go about building a better program.

Another example of such a paper, which uses Science to inform Engineering, is "Representations for Decision-Theoretic Planning: Utility Functions for

Deadline Goals," by Peter Haddawy and Steve Hanks (pp. 71–82). This work explores the problem of combining decision theory and planning, both fields with both practical and abstract incarnations. Haddawy and Hanks define several relationships between decision theory and planning. This is essentially basic Science, albeit Science in which the KR community is tremendously interested. This paper is exemplary, however, in that it ultimately uses these relationships to describe a way of merging the two fields which results in a satisfying planning algorithm. Thus, it addresses both the Scientific challenge of elucidating the relationships between decision theory and planning and the Engineering problem of how to put those insights into practice to build a better planner.

There are many other papers from the KR conference which belong in this section: papers which tell you about what's out there, what exists, and how to characterize it, but which also give you insight into how you can go about building an implementation. Space considerations limit my coverage here; I recommend the conference proceedings for much enjoyable reading.

Before turning to Engineering, though, I would like to point out two other items which belong in the Science category and which are things that I believe have some interest for people outside of KR Science. One was the invited talk by Ray Reiter, called "Twelve Years of Nonmonotonic Reasoning Research: Where (and What) Is the Beef?" (p. 789) That talk, while clearly in the dreaded category of "logic hacking," specifically addressed ways in which the logic hacking influences our practice of KR Engineering.

The second item was an invited talk by Steve Kosslyn, a cognitive psychologist, about Science in the brain: How is it that our brains actually *do* representation and reasoning? It is my belief that he was brought to the KR conference both to remind us that logic and symbolic reasoning are not the only representations—ultimately, we must turn to biology—and to show us that representation and reasoning systems *can* work. Kosslyn's talk was intended in part to help us remember that our goals are KR *systems*, systems which not only have abstractly desirable properties but also are Engineering artifacts that we can use.

3 Engineering

Other papers in the KR conference addressed the problem of Engineering. These papers talk about implemented systems; unlike the familiar characature of a KR paper, full of Greek, they contain performance profiles of programs. These are empirical, result-oriented papers. Did you know that such things were published in the KR proceedings?

One such paper is "An Empirical Analysis of Optimization Techniques for Terminological Representation Systems, or Making KRIS Get a Move On," by Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi (pp. 270–281). This paper addresses the actual effects of optimizations that were believed to improve the performance of terminological logic programs.[2] Terminological logics are KL-ONE-like systems; they are the knowledge representation systems that you are perhaps most likely to use if you decide, in building another type of AI system, to use off the shelf knowledge representation technology. Prior to this conference, there was a well-established theory addressing the optimization of such systems.

Baader et al. examined the existing theoretical literature on optimization of terminological systems, implemented several variants within their terminological system, and analyzed the effect of these optimizations on the performance of the system. Surprisingly, two of the most promising techniques—techniques that were expected from a theoretical standpoint to improve the performance of the programs—did *not* significantly improve performance. In short, this paper describes those techniques that help and those that don't. It takes theory and brings it into the realm of practice, telling the reader what these results actually mean for the ways in which we build KR systems.

Another example of a KR Engineering paper which compares theoretical expectation with empirical practice is "Total Order vs. Partial Order Planning: Factors Influencing Performance," by Steven Minton, Mark Drummond, John L. Bresina, and Andrew B. Philips (pp. 83–92). This paper examines the commonly-held belief that partial orders are a better working representation for plans than total orders. By analyzing the performance of implemented systems, they demonstrate that this hypothesis holds true even for some highly expressive planning languages. They further characterize the circumstances under which partial ordering is advantageous in terms of problem parameters such as solution density, solution clustering, search strategy, and use of heuristics. Like Baader et al.'s paper, Minton et al. depend on empirical analysis to determine ground truth.

There were perhaps half a dozen papers such papers in the conference: papers which compare empir-

[2] In the time since Baader et al.'s paper was written, this class of systems has become more widely known under the name *description logics*.

ical behavior of systems with predictions from theoretical analysis. A somewhat different kind of empirical paper is Yoav Shoham and Moshe Tennenholz's "Emergent Conventions in Multi-Agent Systems: Initial Experimental Results and Observations" (pp. 225–231). This paper is a somewhat preliminary but decidedly empirical investigation of the dynamics of multi-agent systems. It explores the behavior of a society of agents under a set of evolving conventions. Shoham and Tennenholz demonstrate the effect of parameters such as relative attention to one's own vs. others' behavior, or length of memory, on the time to convergence on a uniform societal convention. In this case, a set of experiments leads to the development of a nascent theory which the authors have gone on to explore further in later work.

4 Closing the Cycle

The sections above have talked about KR papers concerned with Science and with Engineering. The Science papers described in section 2 talk about what's out there, about what actually happens in the world. But, in the case of those particular papers, and also in the case of many papers that were not described here, that Science also has something to say to Engineering about fundamental facts that implemented systems can build upon. And so, while these papers spell out a Science of what exists, it is also a Science that has ramifications for the systems that we build. Similarly, the Engineering papers in section 3, and other similar papers in the proceedings, participate in an Engineering which informs the underlying Science about the relevance and legitimacy of its contributions or one which points the Science towards new questions to explore.

At this point, I'd like to turn to papers which address Science and Engineering together. To my mind, one of the best examples of this—but by no means the only example—is a paper called "'Reducing' CLASSIC to Practice: Knowledge Representation Theory Meets Reality," by Ronald J. Brachman (pp. 247–258). This paper addresses the CLASSIC system, again a KL-ONE derivative knowledge representation system, which was first presented in 1989 as a primarily Scientific contribution. CLASSIC is a scaled down version of the KL-ONE family of languages, cleaned up to be a elegant representation of what the KL-ONE knowledge representation family does and contributes. It was a nice piece of Science.

Brachman's KR'92 paper looks at what happened over the succeeding three years, as the CLASSIC group at Bell Labs took the original system—a Scientific, theoretical contribution—and transformed it from the Common Lisp prototype into a deployed product with extensive use outside of their research group. "Reducing CLASSIC to Practice" describes what happened when a piece of Science was taken to the point where it became Engineering. Brachman describes some of the primarily Engineering lessons his group learned. For example, certain aspects of the original, "pure" CLASSIC system were intractable, though possessed of tractable approximations. In implementing the production version of their system, the CLASSIC group was faced with the decidedly non-Scientific question of whether those tractable approximations would survive the next ten generations of CLASSIC. While in Science, it is sufficient to state that an intractable feature has a tractable heuristic approximation, Engineering requires that the developers commit to backwards compatability.

Similarly, getting the abstraction right—making it possible for the user to walk in and understand the basic building blocks of the system—is critically important in a piece of Engineering, though not particularly relevant to Science. And things like user support—whether there is good documentation, whether there are good case examples, whether there are non-toy case studies—matter a great deal in Engineering but relatively little in Science.

On the other hand are the lessons that "real" CLASSIC holds for Science. These are results that were derived from the Engineering experience—from the experience of making this a deployed application—but which speak not to building a system, but to how our theories work.

Real users have real problems. If the user can't understand it, it is wrong. It doesn't matter whether it's theoretically well founded. For example, Brachman's group discovered that users commonly employed a powerful language mechanism to implement minimum and maximum constraints—e.g., `age` is less than 25—which were absent in the original versions of CLASSIC. Because those features arose often in CLASSIC usage, the group ultimately revised not only their implementation but also the underlying theory to handle these particular cases more cleanly. This is an example of something that was learned in Engineering, but changed the Science.

Similarly, engineering the application forced the identification and closure of gaps in the Science on which "pure" CLASSIC was based. For example, "pure" CLASSIC contained a `close` operator over predicates; the implementation revealed that `close` in fact makes sense only as an operator over the entire database. This is a case in which a flaw in the original Science of CLASSIC was discoverd not by further

Science but by Engineering: building the system.

This leads to what I believe is one of the morals of Brachman's paper. To misuse an axiom of Reactive Robotics, theory is doomed to succeed. If you build an artifact, if you prove it useful by implementing it and giving it to a user who has never seen the system before, if that user finds your system helpful, then you have closed the gap. In fact, it may not be until the thirty thousandth satisfied user has proclaimed your system functional that you believe that you have indeed closed the gap. Which means that in order to build a truly useful piece of theory, from a Scientific perspective, you need not only to come up with theory, but to put it in practice, back to theory, and so on and so on and so on: the cycle does not end.

But, don't listen to me, listen to Brachman himself:

> ...the resulting system is clearly far better than anything we could have built in a research vacuum. And the effort of reducing our ideas to a practical system generated a great deal of research...on language constructs, complexity, and even formal semantics....
> [p. 257]

By this, I believe that that he means that not just the implementation but the theoretical, Scientific, abstract version of CLASSIC is clearly better than anything that would have happened if CLASSIC had remained exclusively a piece of Science. This is not to say that Science isn't worthwhile, but rather that Science needs Engineering. And further, in the process of putting into practice this Science, a great deal of truly scientifically interesting research was generated simply by the fact of implementation. So, Engineering informs Science.

"Managing Disjunction for Practical Temporal Reasoning," by Robert Schrag, Mark Boddy, and Jim Carciofini (pp. 36–46), goes the other way. This is a paper which is really about a system. But it is a system that is well founded. It is based on some known theory. This paper addresses the extension of an implementation of Tom Dean's Time Map Manager to cover ambiguity. Dean implemented a system which is essentially a repository for temporal information. Schrag, Boddy, and Carciofini are using it in a scheduling application, but it is relevant to a variety of applications in which temporal information must be stored as well as resolved. Because it is an implemented system, ambiguity is not merely a theoretical problem about which theorems can be proved: it becomes a significant computational problem. Something must be done to resolve ambiguity because the Time Map Manager must support computation to produce useful information; ambiguity becomes an Engineering problem.

The kind of ambiguity that Schrag et al. address is the disjunction which arises when the constraints given to the system leave open multiple possibilities. For example, if I arrive at a meeting at 11:00 am, that may be either before or after the coffee break, which occured at some time between 10:00 am and noon. The system that Schrag et al. built to reason with such constraints is computationally tractable, but it is also theoretically well founded and well characterized. Here are people who are building an Engineering system but were informed by, and informed, Science.

Their solution largely involves limiting the expressive power of the system and substituting a weak approximation to the disjunction. Weak approximation is a well known technique and Engineers use it all the time, but it is something that theory is skittish of. Schrag et al. point out that, in fact, weak representation—approximating the disjunction by an easier to manage, if not, strictly speaking, complete, representation—actually turns out to be a good pragmatic solution. They further close the cycle by demonstrating that their system preserves soundness. And so, Engineering, well founded on theory, with strong ties to a theoretical underpinning, again ties together the cycle of Engineering and Science.

Another example of the interactions of Science and Engineering is "An Approach to Planning with Incomplete Information," by Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Leash, and Mike Williamson (pp. 115–125). This is more of a Science paper—it does not have an implemented system directly tied to it—but it grows out of a problem encountered in constructing a particular implemented system. The paper addresses what seems like a purely theoretical problem: Say that I want to find the blue door. Formally, my goal is to achieve "the blue door." Etzioni et al. point out that this leaves two possibilities: One is that I want some particular door to be(come) blue. The other is that, for example, someone has told me that my jacket is behind the blue door. This means that there are two different kinds of actions that I can perform, both of which will result in my binding "blue door": one is painting the door blue, and the other is looking at the door to see what color it is. In the first case, where I really want this door to be blue, painting it may be appropriate. But in the case where I want to *find* the blue door, painting this door is actually going to interfere in my ability to find the blue door behind which my jacket is hidden. Previous planning languages did not distinguish these two forms of the "blue door" goal.

Etzioni et al. describe a representation language which captures the distinction in these different types of goals: information goals, such as "go out and look and see if the door is blue," vs. goals of achievement, "paint the door blue." Their paper presents the theory behind this, as Science, describing the phenomenological differences between information goals and achievement goals. It culminates in a language within which it is possible to describe conditional plans, an incomplete world model, and the representation of an action upon changing worlds and changing knowledge. But outside of the paper is the implementation which drove the need to discover, or to create, this language. The conclusions of the paper are Science, but its central question was informed by Engineering needs, and having done this Science, Etzioni et al. now have a better idea of how to implement their Engineering system.

The last paper that I'd like to describe is "Towards the Systematic Development of Description Logic Reasoners: CLASP Reconstructed," by Alex Borgida (pp. 259–269). This paper discusses a re-implementation of CLASP, a terminological logic system originally designed and built by Prem Devanbu and Diane Litman. Borgida's interest was in using PROTO-DL, his description language for terminological logics, to implement Devanbu and Litman's already-existing language in a way that uses more advanced software engineering techniques. In the process of doing so, of re-implementing it to meet some software engineering—pragmatic—constraints, he discovered that there were some difficulties in the CLASP language. The theory of PROTO-DL told Borgida how to implement a better system. His new system has software engineering properties which are advantages over the original CLASP system. But the re-implementation, according to those theoretical principles, also informs the way in which CLASP behaves, so that Borgida was able to identify some shortcomings in the terminological logic theory underlying CLASP. Science and software engineering informs practice of reimplementing the system, which in turn informs science on how the system should work in the first place. And the cycle goes on.

5 Keeping the Cycle Turning

I'd like to summarize now the changes that I've seen in the KR community. While there certainly are a lot of papers that might be classified as ivory tower Science—nonmonotonic logic hacking—there are also a lot of papers that have a great deal to say about how Science can inform Engineering, how to build a better system, how to build a system that actually accomplishes your purposes. And there are also a lot of papers that talk about systems that were actually implemented and that have something to say about where Science has missed things, or about where Science should look next. There are Science papers informing Engineering and there are Engineering papers informing Science. And there are also papers that talk explicitly about this cycle. There are papers that say that Science is best when it has something to say to Engineering and Engineering is best when it has something to say to Science.

If this article has a take-home message, it is this: If you consider yourself to be a scientist, look at Engineering. Look at the way that your work can influence Engineering, but also look at Engineering to see what it can say about your work. And if you're an engineer, don't just build your system. Analyze it, or have someone else analyze it. And when you're going out to build your next system, look at what has already been said by Science. See whether that has anything useful to tell you. If you're a KR person, I would ask you to do what the invited talks by Martha Pollack and by Peter Szolovits describe, to look at domains, to look at applications, to take inspiration both about how your system should work—what the world needs—and also about the kinds of problems that Engineering needs you to solve. And if you are not a KR person, I would ask you take another look at KR, to see that the KR community takes this cycle extraordinarily seriously, and to see whether we have solutions to some of your problems, or whether you have problems that we should be solving.