Research Patterns

Graduate students frequently begin research projects with very little background in how to do research. There is a lot of diversity in the topics and styles of research, but there are a variety of patterns in how we do research that can be instructive to beginning researchers. Researchers will have two or three patterns for conducting a research investigation that they use over and over again. The advantage of such patterns is that they can give an overall structure to an investigation. Budding researchers frequently struggle with "what should be done next?", "why do we do it this way?" and "why don't people like my results?". A set of research patterns to start from can help alleviate this problem.

This document outlines a variety of research patterns. It is not intended to be a definitive list nor is it intended to be a careful categorization. It is hoped that that document will grow as other people add successful patterns of their own.

All research consists of a problem to be solved and a proposed or a completed solution to the problem. Research proposals, papers, theses and dissertations consist of problems, solutions and claims about those problems and solutions. Most research patterns revolve around the kinds of claims that are made and the ways to validate such claims. Claims fall into two important categories. The first is the set of claims about the importance of the problem. Unimportant problems are not worthy of research and certainly not worth reading about. The second set of claims is about the value of the solution and claims about its superiority over other solutions.

A well-documented research pattern should have the following components.

- A description of the kinds of claims being made and validated
- A set of strategies to validate claims. These strategies may also include a sequence of research steps required to execute the strategy.
- Examples of good ways to validate the claims. This may include short examples or possibly pointers to papers or other works that provide examples or skills.
- Examples of fallacious support for claims. Where are the common pitfalls in executing the strategy?

Problem Importance Claims

We tend to frequently skip this class of claims. This is particularly true for many of our M.S. students. The most common student fallacy is "This is important because Dr. XXX said it was." This may get you graduated, but it is not good education or good research.

Leverage

Claims

My solution can be applied to a large number of problems.

This is a very important computer science claim. It is, in fact, a key claim when differentiating computers science research from applications programming. Making the claim that solving a problem will make a host of other problems simpler is at the heart of what computers are for.

Strategies for Validation

The key here is demonstrating that the solution really does solve a diverse set of problems. The most important step is identifying the set of problems to be solved. In some cases this can be done formally, in the sense that "My algorithm can parse any context-free grammar." The problem space is well defined and formal proofs can be used to verify your coverage of that space.

Formal definition of a problem space is frequently not possible due to the complexity of the space. The next strategy is to identify a number of example problems from diverse applications and show how the examples have some common feature that is an instance of your problem. The diversity of the examples is helpful to demonstrate the breadth of your solution. The generality argument from these examples must be made. Are the N examples that you present simply N isolated points in the problem space or do they define some region of the problem space that is occupied by many useful problems?

An alternative strategy is the embedded system argument. "If my security solution were embedded in the file system, then every program that uses the file system would automatically be secure." "If I solve the usability problem for tree editing components and embed it into Windows, then every Windows program that uses trees will automatically be more usable." This strategy requires that the embedding be clearly demonstrated and that little or no work is required for other applications to benefit. The more work required for other systems to benefit from your solution, the weaker the leverage argument becomes.

Fallacies

The selected set used to demonstrate diversity is not very diverse, not realistic or very contrived. It is possible to cover a broad and yet unimportant problem space.

Foundational

Claims

My solution is a possible key to a more global problem or a set of problems.
This is an important claim if it is justified. In many problems there is some key block that is preventing further progress. Removing the block can open up further areas of research. Any problem whose solution can open new opportunities is important. The definition of a Turing Machine and the proof that all computing could be encoded as a Turing Machine was foundational in the sense that proofs about Turing Machines were proofs about any program.

Strategies for Validation

One must show that the global problem is actual important and then one must show that solving your problem is indeed a key to the global problem.

A variation on this pattern is a contribution to the global problem. In this case the claim is not that your problem must be solved before the global problem is solved but rather that by solving a portion of the problem, we can make significant progress. There are two ways to do this.

First is to isolate a subset of the problem domain and solve the problem for that subset. For example, linear time parsing of context-free grammars was a longstanding research problem. LL(1) grammars are a subset of context-free but linear-time parsers were developed for the subset. This was extended to LR(k), which turned out to be a very hard subset in its own right, but then SLR(k) and LALR(1) grammars were identified as subsets of LR(k) for which linear-time parsers were developed. In the end the set of grammars not covered by LALR(1) was not deemed important in compiler construction and the work stopped. The key was that each of these subsets of the context-free parsing problem was shown to be important in its own right and a contribution to solving the overall problem.

A second approach is to break down the global problem into component pieces of the solution and then solve one or more of the pieces. The key here is to show that the selected component is indeed necessary to at least one viable solution to the problem and to show that the component is not itself trivially obvious.

Fallacies

A problem of historical importance is not necessarily still important. Text editors that eliminate syntax errors in COBOL programs were once very important. The importance of COBOL has obviated the importance of text editor solutions for it. Moore's law can rapidly remove importance from a problem. Improving an N^2 algorithm to Nln(N) is not interesting any more if N is smaller than 100. Moore's law has eliminated the importance. Such advances are only interesting if N can become very large.

Partitioning off a trivial subproblem. This is where either the solution is obvious, or all the hard work is still standing and nothing of value can be accomplished without addressing the remaining parts of the problem.

Demographic Importance

Claims

My problem is important because some large population has this problem.
This is a common claim in medical research. One might justify the importance of a cure for a disease by quoting statistics about the number of people who have the disease or the number of people who die from it each year.

My problem is important because some significant population has this problem.
The set of people who have this problem are very important or critical people. Examples might be police, firefighters, doctors or national security staff.

Strategies for validation

The key to such claims are to accurately identify the population involved, the importance of that population to the rest of us and the importance of the problem to the population we have identified. Identifying the relevant population can be a problem because for a new technology, nobody owns it and therefore nobody feels like they have the problem. It may be an issue that until we solve the problem, nobody cares.

One approach to making such claims is to identify a related population and then argue how the relationship is valid for this problem. For example, when creating a new home automation system one could argue that people with X10 automation devices installed in their homes would also want this system. One might argue that the population who want our home automation system would be the same size as those who have home theater systems installed. Another example would be a piece of software that manages patient records. The population would be some percentage of medical practices.

There is a wide variety of demographic information available on the Internet. The Census Bureau and the Bureau of Labor Statistics collect lots of information about numbers and kinds of people, what jobs they do and how many are employed in various industries. There are also trade magazine reports on the Internet with sales figures for various products. These are also good demographic indicators of importance.

Fallacies

Every member of population X has this problem. Not all doctors are in private practice and manage patient records. Not all doctors in the world have a patient record requirement. Everyone pays taxes; therefore everyone will want my tax preparation software. Not everyone who pays taxes a) owns a computer, b) has a complicated tax return, c) knows how to use a computer, d) knows how to use the internet to take advantage of my software.

Important people have this problem therefore the problem is important. Surveys may show that most policemen cannot solve rudimentary physics problems but that does not mean that teaching physics to policemen is important. The argument must be completed by showing that physics is important to policemen.

Economic Importance

Claims

My technology will save $999 million per year.

In a capitalistic society money is used to mediate value to the populace. Money is an abstraction for the resources that people possess and price is an abstraction of what people are willing to give for something. There are obvious flaws in the use of money as a measure of value, but capitalistic economies have found it to be the best approximation of value for a variety of goods and services.

Strategies for validation

Research validation for such claims is problematic. The costs of bringing a product to market are frequently detached from the value of the product. There are many other forces at work in the marketplace. Validation of good research must happen long before market forces will have determined a value. The strategy of "see what sells" is fine for product development but not timely for research evaluation.

One way to put economic value on a problem is to measure what might stop if the problem were solved. If I can produce a light bulb that uses 10% less watts than existing bulbs for the same price with the same brightness then I can estimate the cost savings from using my new light technology. The value is estimated based on some current economic activity that will stop when the technology comes into being. This is the argument for the modern office workstation. "N secretaries in the U.S., each paid $Y per hour spend X hours per week retyping drafts of documents. With a word processor the revision time would be cut in half saving $Z per year nationally." This is a justification for the economic importance of word processors.

Fallacies

Underestimating the rate of adoption. Training issues, related products, industry standards and other issues might sharply reduce the economic impact. However, semi-realistic estimates of economic impact are still a good reality check on the importance of a problem and its solution.

Frequently solution costs are underestimated or ignored. In our secretary example, we did not account for the increased costs of a computer over a typewriter. In many situations the technology infrastructure costs outweigh potential savings.

Ignoring interrelationships with other forces, products and technologies. The new low-wattage bulb may not fit standard sockets. Yes it is cheaper, but only after rewiring the house.

Widely Recognized Importance

Claims

My problem is a component of a grand-challenge problem.
The classic instance of this class of claims is proving or disproving P=NP.

Solution Value Claims

We most often concentrate on claims about the value of our solution to a problem. Various kinds of claims carry with them well known strategies for validating those claims. It is these strategies that can form an important framework around which to build our research. Most research problems have claims from more than one of these strategies. Constructing a research plan involves, picking appropriate strategies and adapting their general approaches to the specific needs of the research problem.

Before delving into the strategies themselves there are several strategic fallacies that can occur regardless of the chosen claims.

Solving a subproblem

This occurs when importance is claimed for the research based on some global problem, but the solution does not actually solve the global problem. It solves some portion of the problem, but this remains unstated, or the importance of the actual problem solved is not demonstrated.

Ignoring a relevant claim

An example of this is when an algorithm is developed that necessarily must make some claims to efficiency and yet no such claim is either made or demonstrated. Another example is the creation of a new interactive application where no usability claims are made. Yet another example is when a new methodology is presented to solve a problem without comparing it with existing methodologies to solve the problem. When such a claim should be applied there are several ways to deal with it.

1. Do the additional work to validate the neglected claim
2. Show that the claim is obvious such as “this is equivalent to the sorting problem and therefore can be made Nlog(N)”.
3. Show that the claim does require more work, but that work is not necessary to demonstrate the value of your solution. For example developing a security solution might claim that the value of the solution is independent of the user interface that the vendor places over the solution. Such productization is independent of the core problem. Care must be taken here because a security solution that requires policy to be stated in temporal logic may be completely beyond the skills of its users and therefore ineffective.
4. When many competing views exist, select those deemed most valuable plus those closest to the view being developed and do a careful comparison.

Existence

Claims

Nobody has ever solved this problem and I have a solution.
This is a really good claim if it is true. It is definitely a step forward.

Strategies for Validation

This requires a very careful literature search to make certain that the problem has not been solved. Citations of recent papers that indicate that the problem is an open one are helpful.

Fallacies

Excessive refinement of the problem space. This occurs when a large number of qualifiers are added to a good problem to produce a trivial problem. The real issue is lack of importance. It is true that nobody has done this, but nobody cares. For example, nobody has designed an algorithm for realistically rendering reflections caused by scratches on the surface of silverware, but that doesn't make it an important research problem.

Trivial extension of a problem. Nobody has ever solved this problem underwater. That may be true, but it is either not important to do it underwater or the underwater solution is pretty much the same as other existing solutions with the trivial modification of keeping the computer dry.

Usability

Many solutions are not of any value if nobody can use them.

Claims

My solution is more usable/learnable than other solutions. A key problem here is that usability is not measurable. There are components of usability that can be measured such as time to complete a task, accuracy of the usage, time to learn, attention required, skills transfer, etc. The key to this claim is a comparison to other solutions.

My solution is usable/learnable. The claim here is not that the usability is any better than other solutions but rather that it meets some standard of usability. This is appropriate when the primary claim is something other than usability, such as efficiency or proof of correctness. The point is that my solution should not be rejected due to lack of usability. This claim is also appropriate when this particular problem has not been previously solved (see Existence claims).

Strategies for Validation

The strongest claims are the self-evident claims. That is, the advantages are so clear that user studies are not required. An example of this is the comparison between old line-number oriented text editors modeled after decks of cards and modern full-screen editors. Anyone looking at the two would immediately recognize the usability improvement. Care must be taken, however, in claiming self-evidence. To make this claim, you must show scenarios of usage that show differences of effort of 5-10 times. A second example of this would be comparing a web-browser to the old FTP tools. It is obviously clear that clicking on a link is vastly more usable for most browsing tasks than typing commands into an FTP command stream. This type of validation is only possible when there is a clear paradigm shift in the way the user interacts with the problem. In such cases one can argue in terms of the numbers of commands or interactions required to accomplish the task in the two alternatives.

A second form of the self-evident argument can be used when the claim is only usability not usability improvement. For some problems the fact that someone can actually accomplish the goal with the tool is enough of an argument for usability. In such cases one can argue from the existing skill set of the user population. If the target population consistently uses related tools in substantially the same way, then a usability claim is justified. It must, however, be shown that the population does actually use similar techniques already, not that they "can easily learn them." If this standard does not hold, then more careful validation is required.

When the advantages are not self evident (which they frequently are not) then actual usage of the solution is required as evidence of usability. Measuring usability can be based on metrics such as:

- Time to complete tasks. A representative set of tasks is developed. Each user is given each of the tasks in turn and the time required to complete the task is recorded. More usable solutions will take less time. It is frequently helpful to

create an automated way to evaluate task completion. This simplifies timing and standardizes the evaluation.

- Time to learn the system. A representative set of tasks is developed and a competency standard is established. New users are given training and sample tasks and the number of examples and total time is recorded until the user achieves the competency standard.
- User accuracy. In many tasks the users regularly make mistakes. A more usable solution will reduce the number of mistakes. In other tasks there are levels of accuracy. For example "finding ALL Waldos" might not be as useful as "find as many Waldos as you can in 5 minutes." More usable "Waldo finders" will help the user locate more than less usable ones.
- Learning transfer (see Language/System claims). When one designs a new scroll bar that takes some effort to learn, one may want to demonstrate that the learning effort will pay off in other contexts using the same scroll-bar. The strategy here is to develop two or more independent tests that use the same technology but with different applications. Divide the users into two groups have one group do test A and then test B. Have the other group do test B and then test A. One can then compare first time users of test A to users of test A that have previously experienced your solution in another form. The same comparison can be performed with test B. If the learning transfer hypothesis is valid then there should a marked improvement in other usability measures between first time users and those who have previously learned the solution technology.

The question of usable vs. more usable must be addressed. In the usable case, some standard of performance should be established. In many cases these standards can be quite obvious such as "can complete the task in less than 1 minute" or "can achieve better than 80% accuracy". If the problem is important, then establishing such minimal usability standards can be quite easy.

If there are already solutions to the problem, then the claim must be that your solution is more usable than previously. Where there are substantial paradigm shifts in the way users solve problems, a careful comparison may not be required and only the new solution is tested for usability. Those usage metrics can then be used to make "obviously better" arguments. However, the "obviously better" argument requires that there be huge improvements (2-3 times better with little deviation). If the improvements are not huge, then a one-sided usability test is not valid.

In many cases actual usage comparisons between the new solution and prior solutions must be performed. Usability metrics rarely have absolute scales where new work can be easily compared to other tests performed elsewhere. It is almost always required that solutions be compared under equivalent test conditions. If the measured results of the solutions being compared are at all in the same neighborhood, then tests for statistical significance must be performed.

As a research strategy formative evaluations before actual testing are very helpful. In most cases new solutions have unrecognized usability flaws. The creator of the software

cannot see the flaws because of work and testing habits acquired during development. Before designing any experiments, give the system to anyone at all and video tape their usage. Flaws will start to surface immediately. Do not talk with the subject during the test. Your attempts to justify your design decisions will get in the way of getting the data you need. If they don't understand, then they don't understand and your system or training must be fixed. Several iterations of this process will be required before any experiments for actual usage are warranted. This is also useful for debugging your experimental protocol before bringing in 20 people to get usability data. Documenting a process of finding and correcting flaws can be an important research contribution because it points out false paths that others should not follow.

Fallacies

Inappropriate choice of test subjects. Claims of usability for a broad audience are suspect when all of the test subjects are computer science graduate students working in the same lab. In most situations the test subjects should have equivalent or lower skills/experience than the population for which claims are being made. A frequent problem is too few subjects. Consult a text on statistical significance.

Inadequate controls on the experiment. For a comparison to be valid the only difference between two test groups should be the solution being tested. This includes differences in the order in which tests are given, differences in the way test proctors describe the tasks or provide training. Tasks and training should be provided in written, online or video form so that it is identical from subject to subject. Oral interaction with the subject should be limited so as not to contaminate the results.

Flawed choice of tasks. Picking a set of tasks that is not really representative can skew the results. A particular challenge is picking tasks that are small enough for an experiment and yet complex enough to provide real data. There is a continual tension here that requires careful though and many compromises.

Strawman comparisons. When comparing to other solutions there is a problem of disparate quality. I compare my carefully crafted implementation of my solution against an example of the competing solution that I threw together in 2 days. Details of implementation can make huge usability differences. Using a commercial product or software from some outside source as the comparison can help mitigate such biases. There can also be bias in the quality of the training or instructions given to users of the other system as opposed to your own. Care is required here. There is also the problem of not selecting the best example of the competing solution but rather the ugliest or the most convenient.

Efficiency

Claims

My solution uses less time, memory or other resources than competing solutions.
This is a traditional computer science standard for improvement in research.

My solution uses fewer resources than competing solutions when applied to a subclass of problems.
At this stage of computer science, the most general problems/algorithms are already mapped out. Much research progress is made by restricting the general problem in some way and then producing an algorithm that is more efficient in that subclass.

Strategies for Validation

The formal way to validate such claims is algorithm analysis. Order of complexity analysis should always be done to place the solution in a space of solutions. Worst-case analysis, however, is not always of value. Many algorithms with exponential worst-case complexity behave almost linearly on some problems. Arguing formally about average case, likely case or representative cases can shed light on the claim but is more difficult to do.

Formal analysis should always be augmented by actual measurements. 1) The analysis may be flawed in that some aspect was incorrectly considered. 2) The algorithm may also be too complex for careful analysis with too many tradeoffs between components of the algorithm. 3) On reasonably sized problems the actually coefficients of the complexity terms may matter a lot.

Fallacies

Most of the usability fallacies apply here.

The subclass of problems that I chose to solve is either unimportant or very difficult to describe in a way that anyone can use.

Comparisons across different machines and languages have many problems of timing that are not related to the quality of the compared solutions.

Capacity Improvement

Capacity is the largest problem instance that can be solved using available computational resources using a particular algorithm. Capacity is limited by either time or space. Time-limited capacity means that it takes unacceptably long to solve the problem and space-limited capacity means that there's not enough memory (and/or disk) space to solve the problem. Research in capacity improvement is validated by showing that the new algorithm or implementation solves bigger problems than the old one using the same resources.

Claims

Using the same computational resources, my solution problems that can not be solved by any other solution . Another way of saying that is to say that my solution uses computational resources more efficiently than other solutions on large problems. The pattern for *Efficiency* is similar to this. But capacity claims are targeted specifically at very large problem instances, not average-size or average-workload problem instances. The differences come out in the validation of the claim.

Designing a Claim

0. Identify a problem and algorithm.
1. Identify a capacity limitation in the algorithm. This can be done either empirically or analytically. In most cases, empirical analysis will be needed to at least confirm your hypothesized capacity limitation. Decide if capacity is time or space limited.
2. Create a quantitative model of the algorithm's use of resources that is detailed enough to explain your limitation. This model should begin with a big-O equation, but will be much more detailed. Results from your empirical analysis in step 1 will guide the refinement of this model. "Refinement" means "to add detail".
3. Analyze the model to identify ways to extend capacity. Really and truly, you aren't likely to discover a solution by looking at the model. More likely, you are going to fine tune and better understand your proposed solution by looking at the model. In the worst case, you are just going to use the model to explain your proposed solution.
4. See what other people have done on similar problems. For each paper you read, decide what they are doing to address capacity limitations. In the end of this step, you should have a set of breadth and depth papers. The breadth papers describe a wide range of related work for your problem and algorithm. The depth papers describe work that addresses the same, or similar capacity limitation, and/or use a solution similar to what you are proposing.

Validation

5. Theoretically demonstrate that your solution has some hope of extending capacity using your model from step 2. It may be the case that you have to refine (again, using refine the very narrow sense of adding detail) your model to demonstrate that your solution might work. **At this point, you are ready to propose.**
6. Empirically demonstrate improvement, compared to the existing algorithm, using a suite of benchmarks. Some, if not most, of the problems should be intractable using the existing algorithm on the same computational platform. Consider reading the The Art of Computer Systems Performance Evaluation by R. K. Jain. This is the easiest step to get wrong. A few ways to do this incorrectly are described below in "Fallacies"
7. Revise and extend your model of capacity, from steps 2 and 5, to include your improvement and any other things you discovered in step 6. As the current world expert in your problem and its capacity limitations, use the new model to suggest new research directions for extending capacity even more. **You are ready to defend and new students will use ideas in your step 7 for their step 0.**

Fallacies

Motivate your work talking about space-limited capacity, but then give results that demonstrate improvement of time-limited capacity, or vice-versa.

Justify a capacity improvement based on problems that are solvable using the existing algorithm or implementation.

Improve the efficiency of the implementation or algorithm, but in such a way that does not increase capacity. (This is why modeling is included in steps 2,3, 5 and 7).

Power

Claims

My solution is more powerful than competing solutions

Strategies for Validation

Show by mathematical that the proposed solution solves a larger class of problems. See "Cardinality constraints in semantic data models" in Data and Knowledge Engineering, vol. 11, No. 3, December 1993, 235-270.

Fallacies

More powerful does not mean more useful. If usability is to be tested, employ usability techniques as described in this document.

Developing a Function

Claims

My solution will effectively compute some function.
This claim is found in a large variety of manifestations. This might include locating human skin in an image, translating data into appropriate natural language statements or finding all documents that match some predicate.

My solution can be trained to effectively compute some class of functions.
This is the classical machine-learning claim.

Strategies for Validation

There is a well-defined series of steps involved.

1. Obtain or create a "gold standard" for the function. This is basically a set of data from the problem domain that is clearly associated with the appropriate functional result for that data. In many situations such a gold standard does not exist. It is a very important research step to create an appropriate standard so that research progress can be measured. Identifying and creating a gold standard for some important problem can be a very important contribution in its own right. If a gold standard is not available then.
    a. Create mechanisms for capturing data from the domain where the function is to be applied. The easier the capture, the richer your standard will become.
    b. Create a tool for visualizing data from the functions domain. If you can't see it, you can't work with it.
    c. Augment your tool to allow people to specify appropriate results given example data. This tool will allow you to rapidly create a gold standard from example data. The easier it is to specify the desired function result, the more rapidly you can create the standard and the richer it will be.
    d. Collect and augment data from a broad sample of the domain. Lack of care in this sample will result in less useful results.
2. Define an automated metric for comparing two solution results. In some cases the functional result may be discrete, in which case comparison of results is trivial. However, in many cases there are multiple right answers with varying levels of "goodness". In such cases, carefully defining the metric for comparison is critical to the progress of the research.
3. Build an automated tool that will apply instances of your function to all of the sample data in the gold standard, compare your function's result to the gold standard result and produce some summary measure of the "goodness" of your function. This tool should also highlight where the discrepancies exist between your function and the gold standard. This will rapidly focus your attention to problems that need to be resolved. It should also be easy to exchange instances of your function and compare various solutions that you have developed.

Instructions on the development and testing of trainable solutions should go here.

Fallacies

The domain of the function was not correctly sampled when creating the gold standard. The quality and representative nature of this sample is very important to the quality of the end result and the value of the claims.

The metric for comparison of results is flawed. If the metric is too exact then useful approaches will be or punish inappropriately. If the problem is fuzzy, the metric should capture that fuzziness. The metric may also be flawed if the comparison does not reflect how people actually compare solutions. For example, a metric that compares two sentences based on the number of words they have in common will ignore synonyms and thus incorrectly punish some appropriate translations.

Formal Proof

Claims

My solution has the following properties that I can prove.

Strategies for Validation

A little known secret of proofs in research is that proof is a social process. A proof isn't a proof until a reasonable set of your peers, or reviewers, agree that it's a proof. That said, the strategy for validating a formal proof is to make the proof as understandable and correct as possible. The first step is to pick a formalism in which to describe the theory that leads to your proof. This involves defining notation and terms that are used in the theorem you'd like the prove. The next step is to actually write the theorem down using the notation. Several iterations of designing notation, definitions and theorems may be needed to clearly and concisely describe what you are trying to prove.

Next pick a proof strategy. The structure, or form, of a formally stated conjecture will often demand, or at least suggest, a pattern for validating the proof. In computer science, there are three proof strategies that transcend the form of a conjecture:

1. Reduction to a known class. This would include reducing your problem to an NP-Complete problem or a known decidable problem. The thrust of this claim is "new problem X is decidable if and only if the halting problem is decidable."
2. Transformation to another notation. It may turn out that results from another branch of mathematics or computer science apply nicely to your problem, if you look at your problem just right. For example, a counting problem might reduce nicely to a textbook problem in combinatorics.
3. Structural induction. Arises when you need to prove a property of all sentences in a language. First, give an abstract syntax for the language including atoms and constructors. Next, proof the property on the atoms. Finally, prove the property on the constructors assuming its true on their operands.

Do more detailed proofs than will sometimes be presented. (Detailed proofs are often not palatable and are sometimes too long for page constraints.) This provides assurance that presented high-level proofs have not overlooked subtle detail.

Example: "A Normal Form for Precisely Characterizing Redundancy in Nested Relations" in ACM Transactions on Database Systems, vol. 21, no. 1, March 1996, 77-106.

Fallacies

The properties proved are not interesting or representative of real problems. In pure mathematics, proofs for proofs' sake are appropriate. Computer science is not pure mathematics.

The formal specification of the properties is so obscure that it is not clear that it correctly represents the problem of interest. Even worse, it is not clear to a reviewer what, if anything, you have described or proven.

The notation and definitions are too vague to be considered formally defined. It is not clear to a reviewer what it is you are trying to prove and how you are doing it.

Definitions are harder to write than would be expected. Definitions must be intuitive and must not allow the proofs to become trivial in the sense that they are proved "by definition."

Breadth

Claims

My solution solves a broader class of problems than previous solutions.
This is a very useful way to make progress in a research area.

Strategies for Validation

One must first identify a class of problems for which existing solutions perform poorly. This requires a careful literature search both to identify such a class of problems and to verify that they are not already covered by other work. Next one must show that existing solutions do not acceptably address the chosen class. One must also show that the set of new problems is important. Identifying an example set of problems that is not already solved is crucial. If you can show that my solution solves all of the prior problems plus this new and important additional set of problems, then a claim of progress is justified.

Fallacies

The new set is not important.

My solution does not actually solve all of the problems that the old solutions did and I have not clearly addressed that omission.

Functional Relationship

Claims

These phenomena are related by the following function.
This is classic research in sciences such as chemistry and physics. Its value in computer science lies in discovering such laws when we can and then use to guide the development of new computing solutions.

Strategies for Validation

Fallacies