The inference validity problem in legal discovery*

R. E. K. Stirewalt†, Laura K. Dillon†, and Eileen T. Kraemer*

†Dept. of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA 48824
{stire,ldillon}@cse.msu.edu

*Dept. of Computer Science
The University of Georgia
Athens, Georgia, USA 30332
eileen@cs.uga.edu

Abstract

This paper introduces the inference validity problem, a software-engineering concern that manifests in and complicates the pre-trial process of discovery in litigation. The problem is related to the requirements validation problem in traditional software engineering, but with stricter constraints on stakeholder communication and potentially severe liability risks on the part of software engineers who are retained as expert witnesses. We propose an approach, based on the use of formal methods and traceability, to enable software engineers to avoid this problem, thereby increasing the quality of written opinions while mitigating the risk of liability.

1 Introduction

Discovery is a portion of the pre-trial litigation process during which the parties request from one another information that is relevant to the case. Traditional discovery devices include interrogatories, depositions, and document production requests. During discovery, federal courts and most state courts require each party to disclose any requested information that is reasonably calculated to lead to the discovery of admissible evidence. *Electronic discovery* (*e-discovery*) is a relatively new device that involves securing and searching electronic information, such as corporate e-mail archives or databases. While the device has become indispensable in modern litigation, its use raises a host of knotty issues. In 2006, amendments to the Federal Rules of Civil Procedure [5] went into effect so that the rules of discovery could better accommodate some of these issues—including reasonableness and costs of discovery, preservation of electronic information, form in which information is to be provided, and privacy and security [1, 7]. But current rules of discovery do not address a more fundamental issue, which we call the *inference validity problem* and which falls under the purview of the software engineering profession. This paper introduces and documents this problem and argues that formal methods offer the appropriate tools for solving it.

The inference validity problem arises when e-discovery is used to draw inferences or compute synthetic information rather than to merely collect raw data. The problem occurs both in the case of actual programs written to be executed against a defendant's database and to inferences that are formed after reviewing descriptive materials, such as programmer depositions, selected source code, and documentation. In either case, the expert renders an opinion that contains a database query at some level of abstraction. Simply stated, the inference validity problem concerns whether the query computes what the expert claims it computes and whether the defendant's database could support the query.

Inference validity is ultimately a software engineering concern: A program must be validated against its requirements for its operation to be trusted. Best practices in software engineering support validation through rich and open communication with stakeholders. However, in the context of litigation, communication is severely limited and many of the relevant stakeholders are adversaries. This environment limits the validation tools available to the software engineer; yet the consequences of an invalid inference could incur astronomical costs, especially in class-action suits. In addition, the last few years have seen a dramatic increase in law suits brought by a party against *its own expert* for negligent testimony [2]. As members of an emerging engineering discipline, we believe software engineers should take the lead in defining methods and standards for conducting validation in this context so as to maximize the quality of inference while minimizing the professional risk carried

*This material is based on work supported by LogicBlox Inc, with additional support provided by the National Science Foundation under Grant Number CCF-0702667. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of LogicBlox or the National Science Foundation.

by an individual engineer.

To this end, we are developing a formal method for developing and validating inferences involving e-discovery. Our process derives from Meyer's method of using formalism to structure natural-language specifications [6], which we adapted to use formal modeling and verification tools (in our case Z [9]). The value added by Z is its support for precisely and cleanly:

- modeling the requirements, i.e., to precisely describe the statutory violation that pertains to the complaint,
- modeling the defendant's enterprise database,
- modeling the query (inference), and
- establishing traceability between these models.

The process can be used to produce opinions that are amenable to critical analysis by an opposing expert and that clearly document assumptions, especially party-specific interpretations of statutes. In addition, the logical structure that emerges from formal modeling can be useful for organizing and communicating complex technical ideas to the attorneys who ultimately must argue the case in court and who must depose the expert witness for the other party. Finally, the ability to precisely document assumptions should mitigate the risk of negligence suits.

2 Obstacles to validation

To address the inference-validity problem, a software engineer must validate a proposed inference (i.e., a program) against its requirements. Validation is difficult in the best of circumstances, and this difficulty is exacerbated in the context of litigation. We now explain some of the specific obstacles that arise.

The first obstacle involves the need to precisely analyze the requirements, which includes locating and understanding any statutes that are relevant to the complaint. The typical software engineer will lack the expertise required to know which statutes are relevant. Such a deficiency of domain expertise is common in large software projects and is typically addressed through frequent consultation with domain experts. Unfortunately, in the context of litigation, the domain experts are the attorneys for the parties in the suit. One of these attorneys is an adversary. The other is in a peculiar position that prevents her from actively consulting with an expert witness for fear of being perceived as influencing what should be an impartial opinion based on technical facts. Thus, the software engineer must model and analyze the legal requirements with only limited opportunities for consultation with domain experts.

The second obstacle involves the need for a valid abstract model of the enterprise database itself. This model describes the structure and meaning of the data that are used by a program to draw an inference. It may be significantly more abstract than the "as built" schemas underlying the database[1]; however, for the inference to be valid, the model must be a *sound* abstraction of those schemas. Often, the expert must infer such a model by reviewing selected documents and depositions, and these inferences are prone to error. For instance, given the volume of documentation that an expert must assimilate to form an opinion, he or she could easily craft a model of the database that contains information the real database does not contain. Any query that depends on this unsound data would then be unsound, and any inference based on the query invalid. We know of at least one case, involving WalMart, where the soundness of a database model was used to deny the validity of an inference [10].

It is easy to understand how an engineer could arrive at an unsound model of the database. If not provided with the schemas, she must reverse engineer the model from whatever artifacts are provided or can be acquired through e-discovery. Rarely will such artifacts convey information such as functional dependencies and other integrity constraints. Understanding these constraints is critical when validating an inference that synthesizes data from multiple tables. Moreover, it is easy to assume the existence of a constraint that is not enforced by either the DBMS or the programs that operate on the database. Such assumptions may lead an engineer to conclude that a database contains information that it does not. We suspect this may have caused the soundness problem articulated in Wal-Mart's response to a motion for class certification, which states:

> "the databases analyzed do not contain the requisite information necessary to determine whether and when Wal-Mart's statutory duties were triggered." [10]

The third major obstacle involves the need to trace the requirements outlined by the complaint to elements of the program and/or the data the program computes. To make this more concrete, consider the afore-mentioned case involving Wal-Mart's alleged failure to properly compensate employees following a notice of termination. The relevant federal statutes prescribe the duties of an employer in this situation and may refer to events such as the date when a terminated employee comes to pick up his final paycheck. If such information is not stored explicitly in the database, then an inference must somehow derive it from other information in the database. Such derivations are prone to error and thus require validation.

To summarize, there are essentially three major obstacles to generating a valid inference and/or validating an inference proffered by another party. Overcoming the first

[1] e.g., what is specified via the 'CREATE TABLE' statement in SQL.

two obstacles requires the precise specification of the requirements of the inference and contents of the enterprise database, including functional dependency and integrity constraints. Due to the adversarial nature of litigation, these specification tasks must be conducted with limited stakeholder consultation. Overcoming the third obstacle requires the ability to trace requirements, which refer to concepts in the legal domain, to the data computed by the inference, which refer to concepts in the business domain.

3 Validation using formal methods

Our approach for addressing the inference validity problem borrows from Bertrand Meyer's observations regarding the value of formalism in constructing high-quality, well-organized, natural-language specifications. In a classic paper, he describes how formal modeling can be used to ameliorate a host of validation problems that manifest in what he calls the *seven sins of the specifier* [6]. His thesis is that formalism forces one to structure a complex specification in a manner that naturally avoids the seven sins. Following this idea, our approach advocates the construction of explicit formal models of the requirements and of a faithful abstraction of the database. It also includes a traceability obligation that ensures the validity of the inference itself pending the acceptance of a set of explicitly stated assumptions.

We craft our models in Z, a formal language designed to support conceptual modeling, specification, refinement, and proof [9], and then use the Z models in combination with natural language documentation to systematically address the three obstacles. As a first step, we formally model the violation that the plaintiff claims has occurred as a set of entities that *witness* the violation. In a class-action suit, this set will comprise those individuals with whom the company failed to perform its statutory duties. We refer to this set formally as $Violation$ and introduce it in Z using an axiomatic description, which defines the set in terms of more primitive concepts from the legal domain. For instance, a highly simplified version of the alleged class in the Wal-Mart case might be defined as follows:

$$
\begin{array}{|l}
termDate : Person \pfun Date \\
finalPmtDate : Person \pfun Date \\
Violation : \mathbb{F}\, Person \\
\hline
Violation = \{p : \mathrm{dom}(termDate) \mid \\
\qquad p \notin \mathrm{dom}(finalPmtDate) \lor \\
\qquad termDate(p) \neq finalPmtDate(p)\}
\end{array}
$$

The partial functions $termDate$ and $finalPmtDate$ model, for a given employee, the dates of termination and of final payment respectively. The set $Violation$ contains those employees who were terminated and who did not receive payment on the date of termination.

Next, we model an abstraction of the enterprise database itself using the Z schema language. This model defines the entities, relations, and integrity constraints provided by the database(s). For instance, cash-office databases could be represented using a schema $CashOffice$[2] that tracks cash inflows and outflows at the close of each business day, whereas the regional enterprise database (which records the generation of paychecks) might be modeled as a separate schema $CentralDB$.

Our method prescribes that every element of formalism that appears in a model of the database must be justified with cross-reference to its source in one or more of the review materials. We do this to ensure the soundness of the abstraction. Because Z specifications are traditionally embedded within LaTeX documents, these justifications appear logically very close to their introduction in the model. Indexing into the review materials is simplified by the use of *Bates numbering*, which assigns a unique identifier to every page of every document used in a case. During model construction, the engineer may wish to model data or integrity constraints that he believes must be valid but which cannot be cross-referenced to a source in the review materials. Such assumptions are necessary because review materials are silent on many details that might be relevant to the construction of a sound model of the database. Our method allows for the use of assumptions provided they are documented explicitly as such, thereby allowing an opposing expert to challenge the assumption if needed.

Having modeled the database in Z, we then define the set $Inference$ to contain those individuals computed by the inference. This set is defined within another axiomatic description as a function whose input is the enterprise database. This function is specified using the mathematical toolkit of Z, which contains powerful operators over rich data types. Any inference that can be expressed as a database query (as was the inference in our case) can be specified in this manner, often more concisely. Also, as in the running example, when the enterprise uses multiple distinct databases, we use the Z schema calculus to conjoin these models, adding any relevant join constraints.

Having specified sets $Violation$ and $Inference$, what remains is to argue some kind of containment relationship between the two, e.g., $Inference = Violation$. Because these two sets are defined using concepts from different domains, demonstrating this relation will be the most difficult part of the process. Our method mandates that specification of the inference must be structured in a manner that facilitates traceability in the form of *retrieval invariants*, which show how legal-domain concepts can be retrieved (computed) from business-domain concepts. For instance, we might introduce an invariant that shows, for all employees

[2] definitions for schemas $CashOffice$ and $CentralDB$ are elided for brevity.

$p \in \text{dom}(finalPmtDate)$, that there must exist a disbursement for p on $finalPmtDate(p)$ in either the cash-office or the central database. These invariants represent another kind of assumption and must therefore be formally specified and documented as such. They may then be used as lemmas to prove the containment relationship.

At the end of the process, we have constructed a model of the inference and validated it against a model of the requirements and the database. Any assumptions made in demonstrating validity are explicitly documented and stated in a form that is amenable to challenge by an opposing expert. Each assumption will either be accepted by the other party in the suit or it will be challenged. If all assumptions are accepted by the other party, then that party will have no choice but to accept that the inference is valid, assuming there are no errors in the proof itself. On the other hand, if the other party challenges an assumption, then the attorneys may focus on reconciling these disagreements. This process is generally simpler than arguing over the larger validity of the inference because the explicit statement of working assumptions focuses inquiry.

Regarding benefits to stakeholders, our approach supports a lone engineer, working largely without consultation with domain experts, in the construction of valid inferences and/or in validating the inferences proposed by others. The resulting structure of the inference specification leads to clear explanation to attorneys and judges because it is traceable to the requirements, which are formulated in a domain they understand. Moreover, as argued by Meyer, the resulting structure unfolds complexity in bite-sized chunks and explicitly represents key assumptions, which might otherwise go unstated. In our experience, the organization afforded by this structure greatly simplifies the construction of a written opinion. Finally, this structure offloads much of the risk currently carried by expert witnesses because, if the proof itself is sound, then the inference must hold once all of the assumptions are accepted.

4 Related and future work

Our use of formalism in this domain is most closely related to that of Sergot *et al.*, who used Prolog to codify the British Nationality Act for purposes of validation and exploration [8]. A Prolog program has the benefit of executability, thereby potentially allowing the expert to simulate and pose queries against her models during validation. While successful simulation does not guarantee the validity of models, the technique has proved effective in finding counter-examples [4]. Current tool support for simulating Z specifications is sparse. On the other hand, Z specifications are more modular and readable than large Prolog programs.

To reconcile these competing concerns, we are investigating the application of other conceptual modeling notations and methodologies, specifically Object Role Modeling (ORM) [3]. ORM is supported by a mature modeling and validation environment, called NORMA. We are currently working with the developers of NORMA to produce a Datalog backend, which will allow for the creation and static validation of models using ORM followed by simulation and search for counter-examples using Datalog queries. Such an integrated platform could support the methodology outlined in this paper and could be used to reconcile the claims of competing experts.

References

[1] T. Y. Allman. The impact of the proposed e-discovery rules. *Richmond Journal of Law and Technology*, XII(4):1–25, 2006.

[2] T. Baldas. Hot seat gets hotter for expert witnesses. *The National Law Journal*, May 2008.

[3] T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. Morgan Kaufmann, second edition, 2008.

[4] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[5] Federal rules of civil procedure. Legal Information Institute, Cornell University Law School, 2007. http://www.law.cornell.edu/rules/frcp.

[6] B. Meyer. On formalism in specification. *IEEE Software*, pages 6–26, January 1985.

[7] L. H. Rosenthal. A few thoughts on electronic discovery after December 1, 2006. *The Yale Law Journal Pocket Part*, 116:167–191, 2006.

[8] M. J. Sergot et al. The British Nationality Act as a logic program. *Communications of the ACM*, 29(5):370–386, 1986.

[9] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science. Prentice Hall, New York, 1992.

[10] In Re Wal-Mart Stores, Inc. Wage and Hour Litigation. No. C06-2069, U.S. Dist. LEXIS 14756 (N.D. Cal. Feb. 13, 2008).