Using Production Rules to Aid Security Requirements Acquisition from Legal Texts

Jeremy C. Maxwell

Annie I. Antón

Department of Computer Science

North Carolina State University

{jcmaxwe3, aianton}@ncsu.edu

http://www4.ncsu.edu/~jcmaxwe3/

http://www4.ncsu.edu/~aianton/

Abstract Regulatory compliance is an important consideration for requirements engineering because recent government legislation imposes costly penalties for noncompliance. This paper examines the feasibility of using production rules to model regulatory texts. A production rule model's largest strength is the ability and ease of querying the model for a specific answer to a specific query. This aids requirements engineers in gaining valuable knowledge about legal texts to know what domain knowledge is required to specify the corresponding software requirements. Thus preparing the engineer for formal requirements elicitation sessions with legal domain experts. Additionally, requirements can be validated after specification by querying the production rules model to show concordance with the regulatory model. As an example of production rules modeling, we present a case study based on four sections of the U.S. Heath Insurance Portability and Accountability Act.

Keywords: Requirements engineering, logic programming, regulatory and legal requirements, requirements elicitation

1 Introduction

As recent data breaches show, legal implications of poor data security and privacy can be severe. Fines resulting from violations, cost of court representation and government audits arising from violation of privacy laws can place a significant financial burden on larger companies, and bankrupt smaller ones. The Choicepoint data breach cost the company in excess of 27 million dollars to cover government fines and audits, legal fees, and provide relief to identity theft victims [1]. Providence Health & Services, a Seattle, U.S.A. based health care organization, was fined 100 thousand dollars for violating the United State's Heath Insurance Portability and Accountability Act (HIPAA) Privacy and Security Rules, in addition to the cost of revising company policies, conducting workforce training, and complying to government audits for three years [2]. While a larger health care provider like Providence can handle these costs, a smaller doctor's office or clinic could struggle to meet this financial obligation. Because of the cost of noncompliance, complying with legal regulations must be a focus area for software firms when developing software.

In addition to the high cost of noncompliance, many view regulatory compliance as increasing the information security of an organization. The 2007 Ernst & Young Global Information Security Survey found that 80% of the respondents either agreed or strongly agreed that compliance with government regulations improves security, while only five percent disagreed or strongly disagreed that regulatory compliance improves security. The surveyors also found that regulatory compliance and data privacy and protection were the two top two drivers of information security in the surveyed organizations [3].

Regulatory compliant software, by definition, meets the constraints placed on it by regulations that govern the target environment. Requirements describe the environment of the system to be built [4], so it is important for requirements engineers to examine the appropriate regulations to extract regulatory requirements. Building regulatory compliant software is a difficult problem, however, due to problems with locating relevant regulations, changing regulations, vague interpretations of such legislation, formalization questions, and traceability issues [5]. Governmental regulations such as HIPAA and the European Union's privacy laws [6] require that personally identifiable information be protected, yet extracting requirements from such legal texts is a difficult problem [5]. Thus, requirements engineers need new techniques and tools to aid in specifying and designing compliant software.

In this paper, we examine how production rules can provide a mechanism to assist requirements engineers in specifying security and privacy requirements derived from regulations. Production rules, a knowledge representation technique used in artificial intelligence [7], are usually stated in Horn clauses connected by logical operators [8]. In other words, each rule is an if-then statement. Many such rules combine to create a knowledge base, also called a rules base. To interact with this rules base, a query is presented and viewed as a top-level goal. An inference engine then uses a reasoning strategy, usually backwards chaining, to execute the rules in the rules base. The result is an affirmation or a refutation of the original query [9].

The ability to answer specific queries has been identified as a requirement of any model of legal texts [5, 10]. Querying is particularly useful to requirements engineers unfamiliar with the regulations that govern the target environment. Typically, an engineer would meet with a legal domain expert to gain familiarity with the relevant regulations. However, a requirements engineer might be unprepared to elicit requirements from these legal domain experts because of unfamiliarity with the regulatory text. The ability to receive answers to queries from a production rules model allows the engineer to gain valuable knowledge about the terminology, scope, structure, and substance of a legal text as a pre-elicitation activity. Thus, an engineer can be better informed about what domain knowledge is required to specify the requirements, allowing him or her to elicit regulatory

requirements from legal domain experts more effectively. It is important to note that we are not proposing the replacement of legal domain experts. Instead, we propose the use of production rule models as a tool to aid in acquiring legal requirements from legal domain experts.

Aiding in requirements acquisition is the end use of production rules models. Our work, however, focuses on the process for developing such a model. The case study presented in this paper presents a methodology for translating a regulatory document that governs privacy and security requirements into production rules. The legal text that serves as the subject of our study is the U.S. HIPAA Privacy Rule [11]. Entering full affect in 2004, HIPAA regulates the use of patient records used by health insurance companies, doctor offices, clinics, hospitals, and other organizations. It requires that privacy of patient health records be maintained, except for well-defined circumstances such as treatment by a doctor, filing a medical claim with an insurance company, etc.

One objection that could be raised is the usefulness of production rules models when the U.S. Department of Health and Human Services has released informational fact sheets and fliers [5]. These fact sheets describe the substance of the HIPAA Privacy Rule in simpler language [12], and can be used in a capacity similar to production rule models to provide requirements engineers with domain knowledge. Our work, however, has two advantages over such informational sheets, namely: (1) our work is more comprehensive by covering, in detail, all modeled subsections of the Privacy Rule, whereas the informational flier only provides a summary of the legal text and only considers the most common cases; and, (2) our work provides traceability back to the source subsection in the Privacy Rule.

The remainder of this paper is organized as follows: Section 2 reviews work related to production rule modeling techniques applied to the field of requirements engineering; Section 3 reviews the methodology used in the HIPAA Privacy Rule case study; Section 4 reports the results of converting portions of portions of the HIPAA Privacy Rule into Prolog; Section 5 discusses benefits and challenges of using production rule techniques; Section 6 discusses threats to the validity of the case study; and Section 7 provides a summary of this work and outlines areas of future work in the field.

2 Related Work

Recent work in obtaining regulatory requirements includes the Semantic Parameterization methodology to extract rights and obligations from regulatory texts [13, 14, 15]. This methodology is an implementation of deontic logic, which is concerned with the notions of permission and obligation. In particular, the work by Breaux and Antón [13] provides a foundation for the methodology employed here. Just as their work

classifies regulatory statements into rights, obligations and permissions, the methodology employed here uses production rules to model the rights, obligations, and permissions imposed by the regulation. We also make use of implied rights, obligations and permissions [14], that is, a right held by one group imposes an obligation others.

Knowledge representation techniques proposed for use in requirements engineering include logic, semantic nets, frames and production rules [7]. Production rule modeling of legal domains was a popular area of research in the late eighties and early nineties [16, 17, 18, 19, 20]. In particular, our work is most similar to the ESPLEX project [16]. The land leasing legislation used by Biagioli et al. is a general regulation impacting multiple domains. Similarly, the HIPAA Privacy Rule impacts multiple domains, including health care, law enforcement, the correctional system, and educational institutions. Biagioli et al. also present a methodology for conversion of legal texts to production rules and identify rights, obligations and permissions in the legal text. However, there are several key areas in which our work differs from the ESPLEX project. First, we have a specific goal to analyze production rules models for their feasibility in extracting and understanding regulatory requirements, whereas the ESPLEX project's goal is to help a generic user consult the statute. Second, our methodology differs from the one used in the ESPLEX project. Biagioli et al. first convert the regulatory text to an intermediate form using a normalization process based on the Hohfeld legal concepts. The methodology used in our work does not require an intermediate form, and therefore can be more efficient and accessible, because engineers are not required to have training in the normalization process before translating legal texts into production rules.

The very nature of the HIPAA legislation is different from the legislation used in these works in production rule modeling of legal texts [16, 17, 18, 19]. The legal documents used in these efforts usually seek to answer a single question. For example, in the Supplementary Benefits legislation [17], the goal is to determine if a particular individual is covered under the legislation and therefore entitled to monetary help from the government. In the British Nationality Act [18], the query considered is whether an individual is a British citizen or not. In contrast, the HIPAA Privacy Rule does not have one query that unifies the document. Instead, the broad nature of the legal text allows many possible queries, including queries about access control, the right of notice, requesting restrictions and amendments to health information, interaction with business associates, law enforcement officers, and other external organizations. The broader range of the Privacy Rule makes it difficult to predict potential queries. We choose to use the concepts of rights, obligations, and permissions to capture a broader range of potential queries on the model.

Another work similar to ours is the work by Mitchell et al. on the HIPAA Compliance Checker [21, 22]. They are creating tools usable by health care organizations to enable regulatory compliant expression privacy policies, analyzing and enforcing policies, supporting privacy audits, and discovering inconsistencies in the legal text. Their tool is written in Prolog to check for privacy violations of messages sent to or from a health care organization, and has a web-based front end. A beta version of their present tool is available online[1]. Our work differs from theirs in several respects. First, whereas their goals are varied, we have a narrower focus on assisting requirements elicitation and clarification from legal domain experts. Additionally, the methodology they used to translate the regulation into Prolog is unknown as they have yet to publish a report about this project. We developed a two-activity process to perform this translation, as discussed in Section 3.

3 Methodology

In this section we present concrete examples from our analysis to illustrate how to translate legal texts to production rules to aid in security requirements acquisition. Section 3.1 describes the materials used in our case study. Section 3.2 defines the terminology we will use throughout the translation process. Figure 1 provides an overview of the two activities we employed to perform the translation. The first activity is raw translation of the legal text directly into Prolog using a five step process discussed in Section 3.3. The second activity refactors the rules base to remove duplicate rules and group conditions, and is described in Section 3.4. Translation occurs iteratively; one portion is translated, then the entire rules base is checked for refactoring opportunities.

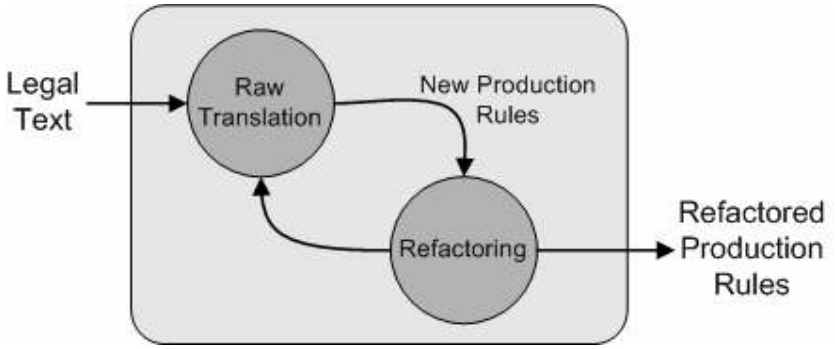

Figure 1. Overview of Translation Activities

[1] http://crypto.stanford.edu/privacy/HIPAA/

We chose Prolog to encode production rules for this paper because of its relatively straightforward design and its prior use in the area of legal knowledge representation. The syntax of a Prolog rule is:

```
<result> :-
     <condition1>,
     <condition2>,
     ...
     <conditionN>.
```

Where the symbol :- is interpreted as the if conditional, the comma symbol is interpreted as logical-and, and the period symbol is interpreted as a full stop (the end of a rule). The result is evaluated to true only if {`condition1, condition2,..., conditionN`} are evaluated to true. The Prolog rule:

```
father(X,Y) :- male(X), child(Y,X).
```

is read "X is the father of Y if X is male and Y is the child of X."

Prolog does have the logical-or construct, but its use is discouraged, because of the branching it causes during execution. Instead, every rule is encouraged to only use logical-and. Logical disjunction has been previously utilized when using Prolog to model legal texts [20], but we have chosen to follow conventions and avoid its use.

In Prolog, an atom is a name, quoted string, or a sequence of special characters (`:-` is one example). A term is the basic unit in Prolog; a term can be an atom, an integer value, a variable or a compound term. A variable signifies a single yet unspecified quantity. A compound term consists of a predicate and its arguments, where a predicate is a relationship between atoms [9]. A predicate that has a single argument is often the assignment of an attribute to the argument. For example, the predicate `isLHCP(doctor)` models the fact that a doctor has the attribute licensed health care professional.

The production rules model makes use of two built-in Prolog commands. The `assert(NewFact)` command adds a new fact to the knowledge base, namely the parameter of the command. Similarly, the `retract(Fact)` command removes the first occurrence of the specified fact from the knowledge base [9].

The strength of Prolog to answer queries comes from two concepts: unification and backtracking [23]. Unification occurs when the inference engine attempts to find a single value to bind to multiple occurrences of a variable. For example, unification would occur if a single value was found for the variable `Org` in the following rule:

```
coveredEntity(Org) :-
     healthPlan(Org).
```

The inference engine uses backtracking to determine the result of a query. The initial query is treated as a top-level goal, then the engine searches the rules base to determine

the value of the goal. For a concrete example, consider a world with only the following facts:

```
man(john).
man(zebulon).
man(johnBoy).
father(zebulon, john).
father(john, johnBoy).
inMilitary(johnBoy).
```

Consider the query:

```
father(X, Y) , inMilitary(Y).
```

This query represents the question "Which fathers have children in the military, and what children would those be?" The inference engine will attempt to first show `father(X, Y)` to be true, then do the same for `inMilitary(Y)`. The engine searches for variables that make the top-level goal, the term `father(X, Y)`, to be true. Two candidates are found, namely `X = zebulon` and `Y = john`. Now the inference engine places the next goal, `inMilitary(Y)`, onto the goal stack. However, `Y` has already been unified to the term `john`, so the fact `inMilitary(john)` is checked. The inference engine fails to verify this fact because this fact does not exist in the rules base, and no rules can show this fact to be true. The engine backtracks to the previous goal, `father(X, Y)`, and attempts to find new values for these variables to satisfy the original query. The new values found for these variables are `X = john` and `Y = johnBoy`. The engine succeeds when it re-attempts the next goal, `inMilitary(Y)`, because the fact `inMilitary(johnBoy)` is in the rules base. Thus, the answer to the original query is John and John-Boy.

3.1 Materials

The materials used in our case study were the same sections of the HIPAA Privacy Rule (§164.520, §164.522, §164.524 and §164.526) that were used by Breaux et al. [14]. We analyzed and converted these sections into Prolog production rules using the SWI-Prolog[2] implementation.

3.2 Terminology

We adopted several conventions to assist us in the translation work. The term *actor* signifies an individual or organization that has a right, is constrained by an obligation, or is permitted to perform some action. Some examples of actors identified during the translation process are shown in Table 1. For naming conventions, noun phrases in the legal text are modeled by a Prolog term bearing that name in mixed case with spaces

[2] http://www.swi-prolog.org/

removed. For example, "correctional institution" is modeled by the term `correctionalInstitution`. The only exception to this rule occurs when a noun phrase is typically an acronym in normal use, and for brevity purposes. For example, "health maintenance organization" is modeled by the Prolog term `hmo`.

Table 1. Sample Actors Identified During Translation

Actor or Object Name in Prolog	Description
`individual`	The individual whose privacy HIPAA protects.
`correctionalInstitution`	A correctional institution.
`hmo`	A health maintenance organization.
`doctor`	A licensed medical doctor.

Building on prior work in regulatory compliance [13, 14], the key query mechanisms are rights, obligations, and permissions of covered entities. These relationships have a loose parallel to faculty (permission), obligation and prohibition previously used in production rule modeling [16].

There are four primary rule types. They are: rights, obligations, permissions, and definitions. Rights, obligations, and permissions all have a pattern we followed when expressing them in Prolog. Rights are encoded in the form:

```
right(A, Y, R, Source)
```

Where the variable `A` signifies the actor that holds the right, the variable `Y` signifies the actor that must honor that right, the variable `R` is actual right, and `Source` is the name of the subsection of HIPAA that specifies this right.

Obligations are actions an actor is legally bound to perform, and are encoded in the form:

```
must(A, O, Source)
```

Where the variable `A` indicates the actor that must carry out the obligation, the variable `O` indicates the name of the obligation, and `Source` is the subsection of HIPAA that specifies this obligation.

Permissions are allowances an actor is provided. That is, an actor *may* perform some action or another. These permissions are encoded in the form:

```
may(A, P, Source)
```

Where the variable `A` indicates the actor that holds the permission, the variable `P` is the actual permission, and `Source` is the subsection of the HIPAA Privacy Rule that specifies this permission.

Definitional rules are the final rule type. These rules do not have a set format like the other rule types. Instead, they typically introduce a new actor or predicate into the rules base. Definitional rules represent a basic domain knowledge that must be translated to Prolog to adequately model the healthcare domain, for example, defining what organizations are considered covered entities or health plans, what is protected health information, etc. Sample definitional rules can be viewed in Figure 2. Many of these rules are derived from definitional sections of the regulation, such as §160.103 and §164.501 of the Privacy Rule.

```
coveredEntity(Org) :-
     healthPlan(Org).
coveredEntity(healthCareProvider).
healthPlan(groupHealthPlan).
healthPlan(hmo).
isPHI(phi).
uses(CE, PHI) :-
     coveredUnderHIPAA(CE, individual, PHI).
```

Figure 2. Sample Definitional Rules

3.3 Raw Translation Activity

Translating regulatory texts into production rules involves five steps, described in this subsection. We followed a systematic procedure that we innovated to express the HIPAA Privacy Rule as Prolog production rules as follows:

1. **Classify rules according to type.** The rule types include: right, obligation, permission or definition. We used normative phrase analysis [14, 24] to determine the type of rule depending on what phrase is present in the legal statement.
2. **Identify rule parameters.** These are the parameters required for the rules types identified in step 1.
3. **Identify preconditions.** Identify the conditions that cause the rule to be true.
4. **Remove rule disjunctions.** Because disjunctions (logical-or) are discouraged, any logical disjunctions in the regulatory text are removed by case splitting the statement into separate rules.
5. **Identify implied rights, obligations and permissions.** Implied obligations and permissions are identified and added to the knowledge base, using right and obligation balancing [14].

The five step process is iterated over every subsection of the legal text to acquire the production rules model. As an illustration, consider the portion of HIPAA Privacy Rule section displayed in Figure 3.

§164.522(a)(1)(i) A covered entity must permit an individual to request that the covered entity restrict:

(A) Uses or disclosures of the protected health information about the individual to carry out treatment, payment, or health care operations;

Figure 3. HIPAA Privacy Rule Excerpt

An application of the raw translation process to Figure 3 is given below:

1. **Classify rules according to type.** We identify this is an obligation by the vocabulary "A covered entity ***must***…".
2. **Identify actors.** The actors in this rule are a covered entity and the individual.
3. **Identify preconditions.** There are several conditions that must hold for this rule to be applied. First, an organization must be a covered entity and the individual must be covered under this covered entity (for example, if the covered entity is a health plan, the individual must be enrolled in the health plan). The information the organization uses or discloses must be classified as protected health information (PHI) under the HIPAA Privacy Rule, and the organization must actually use or disclose the individual's PHI.
4. **Remove rule disjunctions.** To remove disjunctions from the rule, they must be identified and split into separate rules. There are two logical disjunctions in the legal text ("Uses ***or*** disclosures of the protected health information about the individual to carry out treatment, payment, ***or*** health care operations"). Therefore, to split this rule, six Prolog rules are required. Namely, three rules for restriction requests for uses of PHI for treatment, payment and healthcare operations, and three rules for restriction requests for disclosures of PHI for treatment, payment and healthcare operations.
5. **Add implied rights, obligations and permissions** The permission allowing an individual to make a restriction request is an implied permission in this rule.

Twelve Prolog rules were generated as a result of the direct translation to Prolog from the text in Figure 3 using steps 1-5. Figure 4 displays these production rules, numbered for convenience of discussion. Six rules grant an individual rights to restrict uses and disclosure of protected health information for treatment, payment and healthcare operations. An additional six rules grant permission to an individual to request a restriction of the uses and disclosures of that information.

This example shows the need for the refactoring activity. One statement in the legal text translated to twelve Prolog rules. A resulting state space explosion could occur if every legal statement required multiple rules to effectively model. We explain the refactoring activity in Section 3.4.

```
/*1*/right(individual, CE, requests(individual, CE, restrict( uses(CE, PHI) for
     treatment)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     uses(CE, PHI) for treatment,
     isPHI(PHI).
/*2*/right(individual, CE, requests(individual, CE, restrict( uses(CE, PHI) for
     payment)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     uses(CE, PHI) for payment,
     isPHI(PHI).
/*3*/right(individual, CE, requests(individual, CE, restrict(uses( CE, PHI) for
     healthCareOperations)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     uses(CE, PHI) for healthCareOperations,
     isPHI(PHI).
/*4*/right(individual, CE, requests(individual, CE, restrict( discloses(CE, PHI)
     for treatment)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for treatment,
     isPHI(PHI).
/*5*/right(individual, CE, requests(individual, CE, restrict( discloses(CE, PHI)
     for payment)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for payment,
     isPHI(PHI).
/*6*/right(individual, CE, requests(individual, CE, restrict(discloses(CE, PHI)
     for healthCareOperations)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for healthCareOperations,
     isPHI(PHI).
/*7*/may(individual, requests(individual, CE, restrict(uses(CE, PHI) for
     treatment)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for healthCareOperations,
     isPHI(PHI).
/*8*/may(individual, requests(individual, CE, restrict(uses(CE, PHI) for
     payment)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for healthCareOperations,
     isPHI(PHI).
/*9*/may(individual, requests(individual, CE, restrict(uses(CE, PHI) for
     healthCareOperations)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for healthCareOperations,
     isPHI(PHI).
/*10*/may(individual, requests(individual, CE, restrict( discloses(CE, PHI) for
     treatment)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for healthCareOperations,
     isPHI(PHI).
/*11*/may(individual, requests(individual, CE, restrict( discloses(CE, PHI) for
     payment)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for healthCareOperations,
     isPHI(PHI).
```

```
/*12*/may(individual, requests(individual, CE, restrict(discloses( CE, PHI) for
     healthCareOperations)), '164.522(a)(1)(i)(A)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     discloses(CE, PHI) for healthCareOperations,
     isPHI(PHI).
```

Figure 4. Raw Translation of HIPAA §164.522(a)(1)(i)(A)

3.4 Refactoring Activity

During translation, we employed several rule refactoring techniques. The goal of rule refactoring is to identify patterns or similarities in the rules base to reduce rule and condition count. By reducing the number of rules and conditions, we can make the rules base more readable and easier to manage. These techniques may be applied in any order, and should be applied iteratively, and the application of one technique may enable the application of another. Section 4 discusses the impact of refactoring on the rules base.

3.4.1 Refactoring Techniques

Refactor by grouping cases. This refactoring applies when cases, previously split during step 4 of raw translation, can be consolidated, specifically, when the cases consist of multiple items belong in the same set, and the rule is conditioned on one or more of the set members being present. For example, notice the subtle differences in the rules. Rules 1-3 differ only by the purpose of the `uses` condition. Likewise, rules 4-6 differ only by the purpose of the `discloses` condition. Therefore, we can combine these purposes into one set, `valid_HIPAA_tpo`, and modify the conditions to reflect this change. By the introduction of this new predicate, `valid_HIPAA_tpo`, and applying this characteristic to the `treatment`, `payment` and `healthcareOperations` atoms, we reduced the number of required rules. There is some indication that treatment, payment and healthcare operations are meant to be grouped, being called “essential health care functions” [25]. In a similar manner, we recognize in Figure 4 that rules 1 and 4, 2 and 5, and 3 and 6 only differ in the activity the covered entity is using PHI for, namely `uses` or `discloses`. Therefore, we can group the `uses` and `discloses` terms into a new term, `usesOrDiscloses`, that is true when either one of the original terms is true. This reduces the rules count even further. Figure 5 displays the set of completely refactored rules, after all possible refactoring techniques have been applied.

```
/*1*/valid_HIPAA_tpo(treatment).
/*2*/valid_HIPAA_tpo(payment).
/*3*/valid_HIPAA_tpo(healthCareOperations).
/*4*/coveredUnderHIPAA(CE, individual, PHI) :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     isPHI(PHI).
/*5*/usesOrDiscloses(uses(CE, PHI), CE, PHI) :-
     coveredUnderHIPAA(CE, individual, PHI),
     uses(CE, PHI).
/*6*/usesOrDiscloses(discloses(CE, PHI), CE, PHI) :-
     coveredUnderHIPAA(CE, individual, PHI),
     discloses(CE, PHI).


/*7*/right(individual, CE, requests(individual, CE, restrict(Activity for
     Purpose)), '164.522(a)(1)(i)(A)') :-
     valid_HIPAA_tpo(Purpose),
     coveredUnderHIPAA(CE, individual, PHI),
     usesOrDiscloses(Activity, CE, PHI),
     Activity for Purpose.
/*8*/may(individual, requests(individual, CE, restrict(Activity for Purpose)),
     '164.522(a)(1)(i)(A)') :-
     valid_HIPAA_tpo(Purpose),
     coveredUnderHIPAA(CE, individual, PHI),
     usesOrDiscloses(Activity, CE, PHI),
     Activity for Purpose.
```

Figure 5. Prolog Rules After Complete Refactoring

Group common conditions. This refactoring technique groups common rule conditions. If a section in the regulatory text produces multiple rules, all with the same conditions, then a new term can be introduced that is true if the common conditions are true. For example, in Figure 4, many of the Prolog rules have the conditions

```
coveredEntity(CE),
coveredUnder(individual, CE),
isPHI(PHI)
```

in common. We introduce a new predicate, `coveredUnderHIPAA`, that is true if these three conditions are true. We replace any reference to these three conditions with the new predicate `coveredUnderHIPAA`. Even though a new rule is added to the rules base, the overall condition count decreases, especially if many of the rules contain the common conditions. Figure 5 displays the result of applying this refactoring technique.

Replace implied rights, obligations and permissions with actual rights, obligations and permissions wherever possible. Consider the following Privacy Rule excerpt:

§164.524(a)(1) Except as otherwise provided in paragraph (a)(2) or (a)(3) of this section, an individual has a right of access to inspect and obtain a copy of protected health information about the individual in a designated record set.

This section expresses an individual's right to request access to his or her PHI. This implies a covered entity must allow such requests to be made. Therefore, when

performing a raw translation of this section, a Prolog rule expressing this implied obligation will be created. Consider the next subsection:

§164.524(b)(1) The covered entity must permit an individual to request access to inspect or to obtain a copy of the protected health information about the individual that is maintained in a designated record set.

The previously identified implied obligation is explicitly stated here. The original implied obligation associated with §164.524(a)(1) is now replaced with the actual obligation from §164.524(b)(1). This refactoring technique is important because, ideally, each right, obligation, and permission should be traceable to an actual regulation statement rather than an implied one.

Name complex compound terms to improve their readability. This refactoring technique is similar to the *group common conditions* technique. However, instead of introducing new predicates for common conditions, we introduce new predicates for common compound terms. The legal text presented in Figure 6 lists a condition under which a covered entity has permission to deny an individual's request for access to his or her PHI.

§164.524(a)(3) *Reviewable grounds for denial*. A covered entity may deny an individual provided that the individual is given a right to have such denials reviewed, as required by paragraph (a)(4) of this section, in the following circumstances.

(i) A licensed health care professional has determined, in the exercise of professional judgment, that the access requested is reasonably likely to endanger the life or physical safety of the individual or another person.

Figure 6. §164.524(a)(3) of the Privacy Rule

The denial of the request can be represented in Prolog by the fact:

```
denies(CE, request(individual, CE, receive(individual, CE, PHI))).
```

Because of the length of this compound term, it may be awkward to use in other rules as an argument or condition. To make the rules base more readable, a complicated compound term can be replaced with a shorter predicate. For example, we introduce the predicate `accessDenial(CE, individual, PHI)` to replace `denies(CE, request(individual, CE, receive(individual, CE, PHI)))`. This is most useful when the complex compound term appears in many rules. For instance, the new predicate `accessDenial` can be used in §164.524(a)(4), §164.524(b)(2)(i)(B), and §164.524(d) of

the Privacy Rule. To perform the replacement, we add the following rule to the rules base:

```
denies(CE, receive(individual, CE, PHI)) :-
        accessDenial(CE, individual, PHI).
```

We can now replace the longer `denies` term with the shorter `accessDenial` predicate. We condition the original compound term by the original term to ensure that no information is lost through the replacement of the original term.

3.4.2 Avoiding Over-Refactoring

When to stop refactoring is an important question to consider. For example, consider §164.526(f) of the Privacy Rule, which discusses required documentation for PHI amendment requests:

> **§164.526(f)** A covered entity must document the titles of the persons or offices responsible for receiving and processing requests for amendments by individuals and retain the documentation as required by §164.530(j).

Prolog rules created during the raw translation activity for this section are listed in Figure 7.

```
must(CE, document(Title) accordingTo '164.530(j)', '164.526(f)') :-
     coveredEntity(CE),
     is164_526f_title(Title).


is164_526f_title(Title) :-
     title(Title, Person),
     person(Person),
     responsible(Person, receiving(requests(individual, CE, amends(CE, PHI)))),
     responsible(Person, processing(requests(individual, CE, amends(CE, PHI)))),
     isPHI(PHI).


is164_526f_title(Title) :-
     title(Title, Office),
     office(Office),
     responsible(Office, receiving(requests(individual, CE, amends(CE, PHI)))),
     responsible(Office, processing(requests(individual, CE, amends(CE, PHI)))),
     isPHI(PHI).
```

Figure 7. Rules for §164.526(f)

The title and office predicates were not referenced by any other sections of the Privacy Rule that we modeled. Therefore, one may be tempted to refactor the rules in Figure 7 by combining the title into a single Prolog term:

```
must(CE, document('Titles of person or office responsible for receiving
     and processing amendment requests' accordingTo '164.530(j)',
     '164.526(f)') :-
     coveredEntity(CE).
```

Because Prolog predicates used in the raw translation (i.e. `title`, `office`, etc.) are not referenced by any other sections that we modeled, this may seem to be a valid refactoring technique. However, this is an example of over-refactoring. While using the previously mentioned refactoring techniques, we retained the original predicates used in the raw translation; we simply moved them to a separate rule, introduced a new predicate to group common conditions, etc. We over-refactor when reasoning power is removed from the model by removing predicates from the rules base entirely. Rules must not be refactored further if application of a refactoring technique would cause the model to lose expressiveness or meaning. This opens the door for non-compliance. For example, an external legal text might cross-reference this section of the Privacy Rule, or it might be possible for the covered entity's documentation to be incomplete. Therefore, the stopping criteria for refactoring is: if refactoring a given rule (or set of rules) opens the door for potential non-compliance, then do not refactor the rule.

3.5 Keyword Discovery

During the translation process, a number of keywords were discovered that were required to model certain properties or relationships. Prolog predicates were created to model these keywords. This set of keywords (shown in Table 2) is reminiscent of the common privacy policy keywords discovered by Antón et al. [26].

Table 2. List of Relational Keywords

acknowledges	agrees	at	denies
determines	discloses	discusses	documents
enrolled	for	grant	in
informs	intends	knows	maintains
permits	provides	receives	requests
restrict	states	through	to

We expect to discover more keywords as more of the HIPAA Privacy Rule and other regulations are modeled. Breaux and Antón's catalog of constraint categories on rights, obligations and refrainments [13] often parallels the keyword list in Table 2. They review different levels of legal, medical and personal belief determinations, which are indicated by the *determines*, *intends*, and *knows* keywords. Contractual constraints parallel multiple keywords, including *through, states, informs* and *agrees*; the purpose constraints are indicated by the *for* and *to* keywords. Additionally, this list of relational keywords can serve as an initial list for a data glossary or dictionary as proposed in [5].

4 Results of Translation to Prolog

We converted sections §164.520, §164.522, §164.524 and §164.526 of the HIPAA Privacy Rule into Prolog using the methodology introduced in Section 3. The translation process occurred in two phases. During the initial phase, we analyzed the legal test and created a prototypical translation of selected portions of the Privacy Rule to Prolog. From this initial phase, we developed the methodology presented in Section 3, and gained insight on challenging portions of the legal text, which we will review later in this section. During the second phase, we applied the methodology to translate the four sections to create a production rules model. The first phase took approximately took approximately 35 person hours to accomplish, while the second phase took approximately 30 person hours.

Before refactoring, we translated the four sections of the Privacy Rule into 265 production rules, with an average of 3.08 conditions per rule. The rules count reduced to 241 after refactoring, with an average of 2.39 conditions per rule. Table 3 displays the number of production rules, broken down by section, and Table 4 displays the average number of conditions per rule, again broken down by section. These counts are for non-definitional rules, definitional rules will be discussed later in this section.

Table 3. Non-Definitional Rule Count Comparison

	Before Refactoring	After Refactoring
§164.520 Rules	109	99
§164.522 Rules	49	36
§164.524 Rules	58	57
§164.526 Rules	49	49
Total Non-Definitional Rule Count	265	241

Table 4. Average Number of Conditions per Non-Definitional Rule Comparison

	Before Refactoring	After Refactoring
Avg. Conditions per §164.520 Rule	2.95	2.53
Avg. Conditions per §164.522 Rule	3.16	2.50
Avg. Conditions per §164.524 Rule	3.06	2.18
Avg. Conditions per §164.526 Rule	3.27	2.27
Avg. Conditions per Rule (All Sections)	3.08	2.39

Applying the refactoring techniques eliminated 24 rules, or 9.0% of the unrefactored rules base, and reduced the average number of conditions per rule from 3.08 to 2.39, a reduction of 22.4%. It was our experience, however, that a significant portion of the average conditions per rule reduction came from the introduction of a single predicate,

`coveredUnderHIPAA`, as described in Section 3.4. We noticed that we applied this particular predicate less often in section §164.520, so the more modest 13.7% reduction in the average conditions per rule might be a more accurate value.

The anomaly in Tables 3 and 4 is §164.520 of the Privacy Rule, as this section resulted in nearly twice as many production rules as the next highest section, and accounts for 41% of the total rules count. Section §164.520 is not appreciably longer than the next closes section, §164.524, as displayed in Table 5—there are only 356 more words in §164.520. We would not expect a doubling of the rule count for this word increase.

Upon further examination, we discovered the reason for the high rule count is the topic of §164.520. A subsection of §164.520, namely §164.520(b), describes the content of a notice of uses and disclosures of PHI. This section does not place constraints on a covered entity, but rather on the notice a covered entity must maintain. To model this subsection in Prolog, we used rules of the form `contains(notice, X)`, where `X` is a requirement placed on the notice by the Privacy Rule. We obtained 37 production rules of this form for §164.520 before refactoring, and 32 rules after refactoring. Excluding these rules, there are 72 production rules associated with §164.520 before refactoring, and 67 afterwards. These rule counts are within what we would expect, comparing the length of §164.520 with the other sections.

Table 5. Length of Each Section in the HIPAA Privacy Rule

Section	Number of Words
§164.520	2,256
§164.522	556
§164.524	1,903
§164.526	1,253

Table 6 displays a breakdown of the number of rules that follow the right, obligation, and permission patterns, expressed in Prolog with the predicates `right`, `must`, and `may` respectively, as previously discussed in Section 3.2. The rule counts before and after refactoring are presented, as well as the number of rules that do not follow one of the three patterns, grouped into a category called "No Pattern".

Table 6. Rules Partitioned by Pattern

	Before Refactoring				After Refactoring			
Section	**R**	**O**	**P**	**NP**	**R**	**O**	**P**	**NP**
§164.520	9	36	3	61	7	36	3	53
§164.522	7	11	21	10	2	6	13	15
§164.524	3	22	18	15	3	21	17	16
§164.526	2	24	15	8	2	19	12	16
Totals	19	93	57	94	12	82	45	100

Key: (R)ights Pattern, (O)bligation Pattern, (P)ermission Pattern, (NP) No Pattern

The counts in Table 6 reflect the number of *production rules* extracted during the translation process, and not the number of rights, obligations, or permissions found in the Privacy Rule. A single right, obligation, or permission in the Privacy Rule may require several production rules to model properly. It is interesting to note that we discovered fewer rights in the Privacy Rule than discovered by Breaux et al. [14]. This possibly arises from extracting differing amounts of implied rights from the legal text.

Definitional rules are the last set of rules in the model. Definitional rules describe relationships in the Privacy Rule, examples of which can be viewed in Section 3.2, Figure 2. Before refactoring, we made use of 28 definitional rules, derived from the definitional sections of the Privacy Rule, such as §160.103 and §164.501. After refactoring, an additional 12 definitional rules were added to bring the total definitional rules count up to 40. These rules were the result of applying refactoring techniques from Section 3.4 that added new rules to the rules base. For example, we considered the predicates `coveredUnderHIPAA` and `valid_HIPAA_tpo` in Figure 5 to be a definitional rule. Table 7 illustrates the definitional rule counts and average number of conditions per rule, both before and after refactoring. These counts include the rules derived from the definitional sections of the Privacy Rule as well as rules from applying refactoring techniques. If a definition of a term appears in a translated section, however, and that term was only used in the section it is defined in, it was counted as a rule for that section. Several of the sample definitional rules in Figure 2 do not have any conditions—they are simple facts that are unconditionally true. This explains why the average condition per definitional rule count in Table 7 is less than one.

Table 7. Definitional Rule Comparison

	Before Refactoring	After Refactoring
Definitional Rule Count	28	40
Avg. Conditions per Definitional Rule	0.43	0.63

4.1 Sample Model Interaction

To interact with the model, the user executes a series of *assert* statements. These statements are facts that are added to the knowledge base to prepare it for querying. For convenience of discussion, we use the term *situation* to denote a set of assert statement the user employs to set up a particular query. Figure 8 displays a transcript of a sample interaction with the SWI-Prolog model.

```
1 ?- assert(coveredUnder(individual, hmo)).

Yes
2 ?- right(X,Y,Z,S).

X = individual,
Y = hmo,
Z = requests(individual, hmo, restrict(uses(hmo, phi)for treatment)),
S = '164.522(a)(1)(i)(A)' ;

X = individual,
Y = hmo,
Z = requests(individual, hmo, restrict(discloses(hmo, phi)for treatment)),
S = '164.522(a)(1)(i)(A)'

Yes
3 ?- assert(requests(individual, hmo, restrict(discloses(hmo, phi)))).

Yes
4 ?- assert(grant(hmo, requests(individual, hmo, restrict(discloses(hmo, phi))))).

Yes
5 ?- must(hmo,Obligation,Source).

Obligation = not(discloses(hmo, phi)),
Source = '164.522(a)(1)(iii)' ;

Obligation = permit(requests(individual, hmo, receive(individual, hmo, phi))),
Source = '164.524(b)(1)'

Yes
6 ?- retract(coveredUnder(individual, hmo)).

Yes
7 ?- right(X,Y,Z,S).

No
```

Figure 8. Sample Model Execution

Here, three separate situations are queried. The first assert statement in prompt[3] 1 adds the fact to the knowledge base that the individual is covered under a health maintenance organization (HMO). The model responds with the answer `Yes`, meaning the update completed successfully. Prompts 3 and 4 introduce a new situation, where we assert: (a) the individual makes a request for the HMO to restrict disclosures of PHI, and (b) the HMO grants such a request. The third situation is set up in prompt 6 where we retract the assertion the individual is covered under the HMO.

Queries are presented to the model in prompts 2, 5, and 7. The first query in prompt 2 asks the inference engine to determine valid values for each of the variables in the query, which is equivalent to the natural language question "What rights does anyone have, and what sections of the Privacy Rule grant such rights?". The model determines one set of valid values, and lists them, namely, an individual's right to request the HMO to restrict uses of PHI for treatment. The user can prompt the model for another solution by pressing the `;` key (logical-or in Prolog). The inference engine backtracks, looking for another solution, and finds one, namely that an individual has right to request the HMO to restrict

[3] SWI-Prolog uses a command line interface, and the prompts are of the form `N ?-`, where N is a counter that increases with each command entered.

disclosures of PHI for treatment. The user can continue to prompt the more solutions until no more exist, at which point the model would respond `No`. This user can also abort execution by pressing the `a` key, which we did after discovering two solutions.

The query in prompt 5 is executed after the second situation is set up. This query is equivalent to asking the natural language question "What obligations does the HMO have, and what sections of the Privacy Rule impose such obligations?". Again, the model returns a solution, and the user can either repeatedly prompt for other solutions or abort execution. The final query in prompt 7 is executed after the third situation is set up. This third query is the exact same as the first, and the answer is `No`, meaning that there are no rights specified by the Privacy Rule for this situation. This is because the individual is not covered under any covered entity, so the Rule does not apply.

A drawback of this interaction is the high level of familiarity the user must have with the production rules model—the user must know what actors and predicates are available in the model, and how to assert new facts in a meaningful way to be able to query the model. This concern will be discussed in Section 7, where we identify an improved user interface as an area of future work.

5 Discussion

In this section, we discuss several challenges to production rule modeling and regulatory compliance, including traceability, adaptability, ambiguity and complexity of regulations, manual translation, and how to handle temporal conditions.

5.1 Traceability

An important aspect of requirements engineering is traceability, particularly with respect to regulatory requirements. The ability to trace requirements to the originating portion of regulatory law provides the ability to demonstrate due diligence [13].

In our case study, traceability is present in the Prolog rules themselves. We adopted Sherman's solution, adding an additional parameter to each predicate specifying the source of the rule [19]. This is the final field in the `right`, `must`, and `may` predicates presented in Section 3.2. The standard manner of encoding the source is a quoted string of that paragraph name. For instance, if the source of a particular production rule is §164.520(a)(1) of the Privacy Rule, the rule is expressed as:

```
right(X ,Y, Z, '164.520(a)(1)')
```

Extending this, commands can be built into the model that trace the rules applied for a particular query result. Being able to trace the source sections of the legal text facilitates

discussions with legal domain experts, because a requirements engineer can refer to those sections when eliciting requirements from those experts.

Another difficulty in developing any model of legal texts is proving the interpretation used in the model is an accurate interpretation of the law. In discussions with legal domain experts, it might be discovered the interpretation used in the production rules model is incorrect. Traceability allows for correction of the incorrect rules. Another use of traceability is when changes are made to a legal text. The engineer will be able to trace which production rules are impacted by the changes to each section of the legal text and update the rules base accordingly.

5.2 Adaptability

Revisions to regulations are frequent and legal domain experts can reinterpret regulatory documents using case law. Schild and Herzog adopt the use of meta-rules to handle changes in production rule models [27]. Meta-rules support reasoning about production rules in the knowledge base. A common practice in legislation is to have a general case and a special case [10]: the general case applies for every instance except when some condition is met, in which the special case applies. This can be represented generally as follows:

```
A :- not(B).
B :- <condition>
C :- B.
```

Rule A is the general case and rule C is the special case, which is triggered when rule B—the condition—is satisfied. Meta-rules can model precedence of laws, legal heuristics and case law [27], and implement characteristics that Otto and Antón suggested should be present in any regulatory model, including rule prioritization, rule exceptions and regulatory evolution [5].

5.3 Ambiguity and Complexity of Regulations

Researchers have noted that legal texts contain intentional ambiguity and unintentional ambiguity [5, 13]. Intentional ambiguities are those built into the legal texts that allow generalization of the text [5]. Unintentional ambiguity refers to ambiguity that arises from unclear language or other sources where the ambiguity is inadvertent [28].

Consider §164.524(a)(3)(i) of the Privacy Rule presented in Section 3.4, Figure 6. There are several ambiguities expressed in this subsection that need to be resolved before a system is deployed. It is unclear, for example, what “exercise of professional judgment” or “reasonably likely to endanger” someone’s life or safety actually mean. In addition, it

is unclear what type or how much evidence must be present for a licensed health care professional to render such a judgment.

To address ambiguity, May et al. make the distinction between conditions a computer system can verify and those that it cannot [29]. The subsection of the Privacy Rule presented in Figure 6 contains several conditions that cannot be verified by a computer system. The determination made by a licensed health care professional is one such example. To model unverifiable conditions, May et al. propose solutions to set or look for an environmental flag that this condition must be or has already been fulfilled, or delay execution until the condition can be verified by an authorized individual.

In the case study, the following rule was used when modeling Figure 6:

```
may(CE, reviewable(deny(CE, receive(individual, CE, PHI))),
     '164.524(a)(3)(i)') :-
     coveredEntity(CE),
     coveredUnder(individual, CE),
     isPHI(PHI),
     determines(lhcp, 'Likely to endanger individual or others').
```

In the model, a relationship between the licensed health care professional (denoted by the atom `lhcp`) and the determination made by the licensed health care professional evaluates to either true or false. Thus, the ambiguity must be resolved externally and encoded in the model by establishing the appropriate relationship; for example, to establish the relationship, an assert statement could add a fact to the knowledge base that the health care professional has made the determination. To establish a determination has not been made, an assert statement can add a fact explicitly stating this, or no assertion has to be made at all—the Prolog inference engine assumes a fact is false if there is no evidence in the knowledge base that it is true.

Another difficult challenge in modeling regulatory texts is the complexity of legal texts. Specifically, frequent cross-referencing to other portions of the text or to separate legislation entirely increases the complexity of a legal document [5, 13]. References to portions of the same legal text are called internal cross-references, while external cross references refer to other legal texts.

The inference engine used to model regulatory texts with production rules aids in checking internal cross-references—this entails invoking the rules associated with the referenced portion. To view an example of cross-referencing, consider portions of sections §164.524(a)(1)-(2)(i) of the HIPAA Privacy Rule, displayed in Figure 9.

§ 164.524 Access of individuals to protected health information.

(a) *Standard: Access to protected health information*.

(1) *Right of access.* Except as otherwise provided in paragraph (a)(2) or (a)(3) of this section, an individual has a right of access to inspect and obtain a copy of protected health information about the individual in a designated record set, for as long as the protected health information is maintained in the designated record set, except for:

(i) Psychotherapy notes;

(ii) Information compiled in reasonable anticipation of, or for use in, a civil, criminal, or administrative action or proceeding; and

(iii) Protected health information maintained by a covered entity that is:

(A) Subject to the Clinical Laboratory Improvements Amendments of 1988, 42 U.S.C. 263a, to the extent the provision of access to the individual would be prohibited by law; or

(B) Exempt from the Clinical Laboratory Improvements Amendments of 1988, pursuant to 42 CFR 493.3(a)(2).

(2) *Unreviewable grounds for denial.* A covered entity may deny an individual access without providing the individual an opportunity for review, in the following circumstances.

(i) The protected health information is excepted from the right of access by paragraph (a)(1) of this section.

Figure 9. Cross References in the Privacy Rule

Both internal cross-referencing and external cross-referencing can be viewed in this section. The internal cross-referencing is present in §164.524(2)(i), where a reference is made back to §164.524(a)(1). This cross-reference is resolved by referencing the rules associated with that section, displayed in Figure 10. The inference engine, when evaluating the rule associated with §164.524(2)(i) (rule 6 in Figure 10), will attempt to resolve `s164_524_a_1_exception(PHI)` as a goal. When resolving this goal, the inference engine will invoke the rules associated with §164.524(a)(1). In this manner, the inference engine can be leveraged to resolve internal cross-references. For the purposes of our case study, internal references that reference portions of the Privacy Rule not included in the model are treated as external cross-references, which we now describe.

```
/*1*/right(individual,CE, receive(individual,CE,PHI), '164.524(a)(1)') :-
     coveredUnderHIPAA(CE, individual, PHI),
     maintains(CE, PHI),
     not(s164_524_a_1_exception(PHI)).


/*2*/s164_524_a_1_exception(psychotherapyNotes).
/*3*/s164_524_a_1_exception(PHI) :-
     PHI for courtProceeding.


/*4*/s164_524_a_1_exception(PHI) :-
     subjectToClinicalLabImprovements1988_42USC_263a(PHI).


/*5*/s164_524_a_1_exception(PHI) :-
     exemptFromClinicalLabImprovements1988_42CFR_493_3a2(PHI).


/*6*/may(X, unreviewable(denies(X, receive(individual, X, PHI))),
     '164.524(a)(2)(i)') :-
     coveredEntity(X),
     coveredUnder(individual, X),
     s164_524_a_1_exception(PHI),
     isPHI(PHI).
```

Figure 10. Cross References in the Production Rules Model

Cross-referencing with external legislation is more difficult than internal cross-referencing. In Figure 9, external cross-references exist in §164.524(a)(1)(iii)(A-B) to the Clinical Laboratory Improvements Amendments of 1988. One could analyze this external legislation, model it using production rules, and insert a reference to that knowledge base in the HIPAA Privacy Rule model. The problem with this solution, however, could be a large number of legal texts that need to the modeled, if the Improvements Amendments referenced some other legal document, that referenced some other legal document, etc. Indeed, this is the method of choice for the most complete and extensive model, but it is out of scope for our case study.

Instead, we treat external cross referencing as a condition the computer system cannot verify, and rely on an environmental variable to track value of the condition. In Figure 10, we use the two predicates `subjectToClinicalLabImprovements1988_42USC_263a(PHI)` and `exemptFromClinicalLabImprovements1988_42CFR_493_3a2(PHI)` in rules 4 and 5, respectively. By default, no rule is present that would cause these predicates to evaluate to true. During execution, however, the user can use assertions to set which predicates are true. In this manner, external cross-referencing must be resolved externally by the user of the model.

5.4 Manual Translation

Manual translation of a regulatory text into any representation is costly because it must be verified. Human error can introduce inconsistencies in the model. Therefore, an expert in the legal domain often must be consulted to verify the correctness of the representation. This only applies in creation of the model, however. Reuse of the model may offset this cost. Regulatory text only needs to be translated once and could be reused by requirements engineers for multiple projects subject to the same regulations. Developing this type of tool-supported process is an area of future research for modeling legal texts using production rules.

5.5 The Issue of Time

Researchers have identified temporality—the ordering of past, present, and future events—as a particularly challenging issue to handle in a production rule model of laws [17, 19]. For example, a legal text can place a limit on the time frame an action may be performed in, illustrated by section §164.524(b)(2)(i) of the HIPAA Privacy Rule. This section requires a covered entity to act on an individual's request for protected health information within 30 days.

We view the production rules model as a snapshot of reality at a particular moment in time. With this view, the model can make no claims about the future nor can it reason about possible future worlds. Instead, our primary concern is the modeling of preconditions and postconditions for use in rule conditions. To model such conditions, we adapted the keywords introduced for capturing time operations [24]. Predicates such as `after` and `within` were utilized for modeling temporal conditions. These predicates were not used to create a total ordering on the events in the model, but instead were strictly used to capture temporal conditions in the legal text.

6 Threats to Validity

In this paper, we describe a methodology to translate a legal text into production rules. We used four sections of the HIPAA Privacy Rule as a cases study to both develop and apply the methodology. Our case study is an exploratory case study, and therefore internal validity is not a concern [30]. Internal validity addresses causal relationships—we make no inferences as the result of our case study, and so internal validity is not applicable. Construct validity, external validity, and reliability do concern our case study, which we now discuss.

6.1 Construct Validity

Construct validity addresses the degree to which a case study is in accordance with the theoretical concepts used [30]. The key concepts we made use of in our case study were production rules, rights, obligation, and permissions. Our use of the term *production rules model* accords with standard uses of the term in the literature (for example [8]). Likewise, we derive our use of the concepts of right, obligation, and permission from their uses in the literature (for example [14]).

6.2 External Validity

External validity addresses the ability of a case study's findings to be generalized to other domains under different settings [30]. We recognize several threats to the external validity of our case study. We only examine one legal text, the HIPAA Privacy Rule, which regulates only one domain, the healthcare industry. Furthermore, we only translated a portion of the Privacy Rule to production rules.

Mitigating these threats, the selected portions of the Privacy Rule exhibit many of the previously identified properties of legal texts [5], including internal and external cross-referencing, domain knowledge of the definitions used, and ambiguous language used in the legal text. Further studies in other domains and with different legal texts will serve to validate and refine the methodology developed here.

6.3 Reliability

Reliability addresses the ability to repeat a case study and reproduce similar results [30]. Researchers repeating our case study would potentially make use of different Prolog predicates, state rules and conditions differently, and identify different patterns to be refactored. Thus, there is very little chance that the *exact* production rules model created in our case study could be replicated in a repeat case study. Application of the two step methodology introduced in Section 3 would indeed produce a production rules model, however. Thus, reliability is improved by measuring reliability by the criterion that some production rules model is produced by application of our methodology, and not measuring reliability by the exact production rules model we created.

7 Summary and Areas of Future Work

This paper presents a methodology to translate legal texts into a production rules model. This includes a two activity process that is applied iteratively on each section of the legal text. First, a raw translation is produced, then the entire rules base is refactored by repeated application of refactoring techniques. A production rules model is useful to requirements engineers by providing access to domain knowledge. An engineer can become familiar with the structure, concepts and terminology used by a legal text by querying a production rules model. An engineer familiar with a legal text can have more efficient requirements elicitation sessions with legal domain experts, and thus be able to more effectively elicit the security and privacy requirements a system must meet to comply with law.

There are several limitations of using production rules models. The querying process is lengthy, and an engineer must know what predicates are available to query. An improved user interface is expected to overcome these limitations. A web-based interface, similar to the work by Mitchell et al. [21, 22] as discussed in Section 2, is a possible solution. Any user interface for a production rules model must contain:

- an indication of the actors in the model,
- a list of predicates that can be used in assertions, and
- documentation for each predicate to indicate to the user the arguments and intended use of the predicate.

Performing a complete translation of the remaining sections of the Privacy Rule and constructing a user interface that meets the above criteria would create a usable tool that allows requirements engineers to use a production rules model of the Privacy Rule effectively.

A study of the utility of production rules models in an actual requirements engineering setting is an important area of future work. In addition, little work has examined the affect of changing legislation on regulatory requirements. We plan to study the affect changes have on a production rules model, in terms of the rules modified, as HIPAA case law develops. In addition, work is needed to study the affects of external cross-references in a legal text. In our work, we rely on environmental flags to resolve cross-references, but this places the onus on the user to check the external legislation for compliance. This is an important area of research that has yet to be considered.

References

[1] Otto, P.N., Antón, A.I., Baumer, D.L., "The Choicepoint Dilemma: How Data Brokers Should Handle the Privacy of Personal Information", *IEEE Security and Privacy*, vol. 5, no. 5, Sep.-Oct. 2007, pp. 15-23.

[2] U.S. Dept. of Health and Human Services, "HHS, Providence Health & Services Agree on Corrective Action Plan to Protect Health Information", News release, July 17, 2008, <http://www.hhs.gov/news/press/ 2008pres/07/20080717a.html>

[3] Ernst & Young, *Global Information Security Survey 2007*, <http://www.ey.com/global/content.nsf/International/AABS_-_TSRS_-_GISS_2007_Request_Form>.

[4] Jackson, M. "The Meaning of Requirements", *Annals of Software Engineering*, vol. 3, Baltzer Science Publishers, 1997, pp. 5-21.

[5] Otto, P.N., and Antón, A.I. "Addressing Legal Requirements in Requirements Engineering", *Proc. of the 15th IEEE International Requirements Engineering Conference*, New Dehli, Oct. 15-19, 2007, pp. 5-14.

[6] Council Directive 95/46/EC, 1995 O.J. (L 281) 31.

[7] Dubois, E., Hagelstein, J., Lahou, E., Ponsaert, F., and Rifaut, A., "A Knowledge Representation Language for Requirements Engineering", *Transactions of the IEEE*, vol. 74, no. 10. Oct. 1986, pp. 1431-1444.

[8] Brachman, R.J., and Levesque, H.J., *Knowledge Representation and Reasoning*, San Francisco: Elsevier, 2004.

[9] Sterling, L., and Shapiro, E., *The Art of Prolog: Advanced Programming Techniques*, Cambridge, Mass.: MIT Press, 1994, 2nd ed.

[10] Antoniou, G., Billington, D. and Maher, M.J., "On Analysis of Regulations using Defeasible Rules", *Proc. of the 32nd Hawaii Intl. Conf. on System Sciences*, 1999, pp. 1-7.

[11] Health Insurance Portability and Accountability Act of 1996, 42 U.S.C.A. 1320d to d-8 (West Supp. 1998).

[12] U.S. Department of Health and Human Services Office of Civil Rights, "Your Health Information Privacy Rights", Informational Flier, <http:// www.hhs.gov/ocr/hipaa/consumer_rights.pdf>

[13] Breaux, T.D. and Antón, A.I., "Analyzing Regulatory Rules for Privacy and Security Requirements", *IEEE Trans. on Software Engineering*, Vol. 34, No. 1, Jan.-Feb. 2008, pp. 5-20.

[14] Breaux, T.D., Vail, M.W., and Antón, A.I., "Towards Regulatory Compliance: Extracting Rights and Obligations to Align Requirements with Regulations", *Proc. of the 14th IEEE Intl. Requirements Engineering Conf.*, Minneapolis, Sep. 11-15, 2006, pp. 46-55.

[15] Breaux, T.D., Antón, A.I., and Doyle, J., "Semantic Parameterization: A Process for Modeling Domain Descriptions", *ACM Trans. on Soft. Eng. Methodologies*, (In Press) 2009.

[16] Biagioli, C., Mariani, P., and Tiscornia, D., "Esplex: A Rule and Conceptual Model for Representing Statutes", *Proc. of the 1st ACM Intl. Conf. on Artificial Intelligence and Law*, Boston, 1987, pp. 240-251.

[17] Bench-Capon, T.J.M., Robinson, G.O., Routen, T.W., and Sergot, M.J., "Logic Programming for Large Scale Applications in Law: A Formalisation of Supplementary Benefit Legislation", *Proc. of the 1st ACM Intl. Conf. on Artificial Intelligence and Law,* Boston, May 1987, pp. 190-198.

[18] Sergot, M.J., Sadri, F., Kowalski, A., Kriwaczek, F., Hammond, P., and Cory, H.T., "The British Nationality Act as a Logic Program", *Comm. of the ACM*, Vol. 29, No. 5, May 1986, pp. 370-386.

[19] Sherman, D.M. "A Prolog Model of the Income Tax Act of Canada", *Proc. of the 1st ACM Intl. Conf. on Artificial Intelligence and Law*, ACM, Boston, May 1987, pp. 127-136.

[20] Sergot, M.J., Kamble, A.S., Bajaj, K.K., "Indian Central Civil Service Pension Rules: A Case Study in Logic Programming Applied to Regulations", *Proc. of the 3rd ACM Intl. Conf. on Artificial Intelligence and Law*, Oxford, 1991, pp. 118-127.

[21] Ho, Anthony, and Sundaram, Sharada, *A Prolog Based HIPAA Online Compliance Auditor*, Unpublished class report, Mar. 20, 2008, <www.stanford.edu/class/cs259/projects/ cs259-final-Sharada%20Sundaram%20Anthony%20 Ho/report.pdf>.

[22] Mitchell, J.C., *Medical Privacy and Business Process Design*, Presentation, Stanford Computer Forum, March 17, 2008, <http://forum.stanford.edu/events/2008slides/Security%20 Workshop%20Slides/John%20Mitchell-forum-workshop-08.pdf>.

[23] Sethi, R., *Programming Languages: Concepts and Constructs*, Reading, Mass.: Addison-Wesley, 1990.

[24] Breaux, T.D., and Antón, A.I., "Mining Rule Semantics to Understand Legislative Compliance", *Proc. of the 2005 ACM Workshop on Privacy in the Electronic Society*, Alexandria, USA, Nov. 7, 2005, pp. 51-54.

[25] United States Dept. of Health and Human Services, *Protected Health Information*, Presentation, 2003, http://www.dhhs.gov/ocr/hipaa/conference/udmn.pdf.

[26] Antón, A. I., Earp, J.B., He., Q., Stufflebeam, S., Bolchini, D., and Jensen, C., "Financial Privacy Policies and the Need for Standardization", *IEEE Security and Privacy*, vol. 2, no. 2, Mar./Apr. 2004, pp. 36-45.

[27] Schild, U.J, and Herzog, S. "The Use of Meta-Rules in Rule Based Legal Computer Systems", *Proc. of the 4th ACM Intl. Conf. on Artificial Intelligence and Law*, Amsterdam, 1993, pp. 100-109.

[28] Layman, L.E., "Symbolic Logic: A Razor-Edge Tool for Drafting and Interpreting Legal Documents", *The Yale Law Journal*, Vol. 66, No. 6, May 1957, pp. 833-879.

[29] May, M.J., Gunter, C.A., and Lee, I., "Privacy API's: Access Control Techniques to Analyze and Verify Legal Privacy Policies", *Proc. of the 19th IEEE Computer Security Foundations Workshop*, Venice, Italy, Jul. 5-7, 2006, 13 pp.

[30] Yin, R.K., *Case Study Research: Design and Methods*, in Applied Social Research Methods Series, Vol. 5, Thousand Oaks, CA: Sage Publications, 2003, 3rd ed.