Rumpole - An Introspective Break-glass Access Control Language

SRDJAN MARINOVIC, Institute of Information Security, ETH Zurich
NARANKER DULAY, Department of Computing, Imperial College London
MORRIS SLOMAN, Department of Computing, Imperial College London

Access control policies define what resources can be accessed by which subjects and under which conditions. It is, however, often not possible to anticipate all subjects that should be permitted access and the conditions under which they should be permitted. For example, predicting and correctly encoding all emergency and exceptional situations is impractical. Traditional access control models simply deny all requests that are not permitted, and in doing so may cause unpredictable and unacceptable consequences. To overcome this issue, break-glass access control models permit a subject to override an access control denial, if he accepts a set of obligatory actions and certain override conditions are met. Existing break-glass models are limited in how the override decision is specified. They either grant overrides for a pre-defined set of exceptional situations, or they grant unlimited overrides to selected subjects, and as such they suffer from the difficulty of correctly encoding and predicting all override situations and permissions. To address this, we develop Rumpole, a novel break-glass language that explicitly represents and infers *knowledge gaps* and *knowledge conflicts* about the subject's attributes and the contextual conditions, such as emergencies. For example, a Rumpole policy can distinguish whether or not it is known that an emergency holds. This leads to a more informed decision for an override request, whereas current break-glass languages simply assume that there is no emergency if the evidence for it is missing. To formally define Rumpole, we construct a novel many-valued logic programming language called Beagle. It has a simple syntax similar to that of Datalog, and its semantics is an extension of Fitting's bilattice-based semantics for logic programs. Beagle is a knowledge non-monotonic langauge, and as such is strictly more expressive than current many-valued logic programming languages.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection

General Terms: Design, Algorithms

Additional Key Words and Phrases: security, access control, break-glass access control, logic programming, many-valued logics

ACM Reference Format:
Marinovic, S., Dulay, N., Sloman, M. 2012. Rumpole - An Introspective Break-glass Access Control Model. ACM Trans. Info. Syst. Sec. X, X, Article XX (July 2013), 31 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Access control policies define which subjects are permitted to use protected resources and under which conditions. A crucial assumption underlying access control enforcement is that policies are complete and precisely define the subject, the resource and the conditions for access. In life-critical domains, such as healthcare [Anderson 1996], it is unrealistic to have a complete set of policies, since it is not practical to anticipate all the situations that may occur that require an access control decision. Furthermore

Author's addresses: S. Marinovic, Institute of Information Security, ETH; N. Dulay and M. Sloman, Department of Computing, Imperial College London.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2013 ACM 1094-9224/2013/07-ARTXX $15.00
DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

©Marinovic, Dulay, Sloman, 2014. This is the authors' version of the work. It is posted here for your personal use. Not for redistribution. The definitive work will be published in ACM TISSEC.

the information that is available to make access control decisions may be incomplete, contradictory or unreliable. The gap that is created between resource needs, policy specifications and poor quality information can lead to systems where access control is too strict – preventing critical resources being accessible, for example, in an emergency.

The default behaviour of most access control systems is to deny all requests that are not permitted by a security policy. Even if a subject has a justified reason for requesting access there is no mechanism to override the denial, other then asking a security administrator to change the policy. The subject is never given the benefit of the doubt or allowed to present new evidence for why access should be permitted. We do not argue that all subjects have to be given the benefit of the doubt, but that in some applications the consequences of denial might be more damaging than allowing the override.

Break-glass access control models allow a subject to override access control decisions but impose obligatory actions on the subject and/or the system itself [Povey 2000] [WBG 2004] [Longstaff et al. 2000]. Break-glass policies attempt to mediate between a normal access control policy and the subject by asking the subject to accept specific obligations that offset the perceived risk introduced by the override. We stress that not all override requests need to be granted, some requests may be deemed to risky or inappropriate.

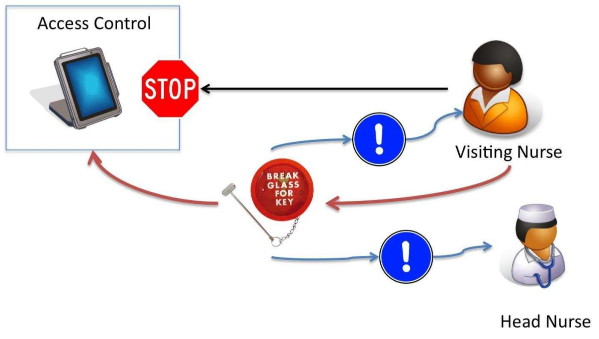

Fig. 1. Overriding an access control denial.

To illustrate the break-glass model consider the healthcare example shown in Figure 1. Late at night, a visiting nurse notices a mild allergic reaction developing on a patient's arm. This is not a high-risk emergency, but it would be considered good practice for the nurse to treat it. She needs some information from the patient's record to select an appropriate treatment, but she is not authorised to treat patients in this ward, especially when the situation is not life threatening. Break-glass policies are able to address such situations. For example, a break-glass policy may grant an override to an appropriately qualified nurse if she accepts to have her actions recorded by the hospital's CCTV cameras and reported to the head-nurse. Such obligations have a two-fold purpose. First, they raise the subject's awareness that the request is unanticipated and potentially has serious consequences. Second they attempt to provide a means to hold subjects accountable for their actions.

Current break-glass models either grant overrides for exceptional situations [Brucker and Petritsch 2009] [Ardagna et al. 2010] [Ardagna et al. 2008] [Gupta et al. 2006] [Longstaff et al. 2000], or they grant unlimited overrides to selected subjects [Ferreira et al. 2009] [Rissanen et al. 2004] [Povey 2000]. Both of these approaches, however, suffer from the difficulty of correctly encoding and predicting all override situations and permissions – the same problem that afflicts traditional access control policies.

Encoding all conditions that determine whether an exceptional situation exists or not, is not feasible. Similarly *a priori* deciding all subjects that should be given override access may also be infeasible. To address these issues, we develop Rumpole, a novel break-glass language that explicitly represents and infers *knowledge gaps* and *knowledge conflicts*. We say that there is a *knowledge gap* when it is not possible to infer, based on the supplied evidence whether a condition holds or not, or whether an override can be granted or denied i.e. uncertainty in the available information. Similarly, a *knowledge conflict* denotes that it is possible to infer that some condition both holds and does not hold. For example, if fire alarms have failed, or if a user is not present to manually signal an emergency, Rumpole's semantics would infer that it is "unknown" whether there is an emergency or not.

When combining evidence from different sources, different levels of confidence in their reliability may be needed. For example, a nurse signalling an emergency can be taken as strong evidence, but temperature sensor data can often produce poor quality information, which might appropriately be labelled as "unreliably-true" or "unreliably-false". Therefore the composition of nurse and sensor evidence for an emergency does not necessarily result in a clear conflict but rather a "weak-conflict". These weaker values also represent gaps and inconsistencies. The range of these unreliable values is domain specific and Rumpole lets break-glass policies specify any discrete range that they require.

Rumpole represents gaps and inconsistencies both at the language (semantic) meta-level and also at the language (syntactic) object-level. We, therefore, refer to Rumpole as an *introspective* language since its rules can be constrained with knowledge levels.

Consider our earlier example. Rumpole can express a break-glass policy, which when it is unknown whether there is an any emergency would ask the nurse to agree to a stricter obligation, such as CCTV and sound recording. A policy could also say that if it is unknown whether a subject has been *blacklisted* by the patient (e.g. such information cannot be obtained at a current moment), and there is conflicting evidence that there is an emergency, then the override can be given provided that a subject agrees that the override will be reviewed and inspected by the head-nurse. It is precisely this additional (epistemic) specification dimension, which we believe leads to more flexible break-glass policies that can cope with a wider range of override requests.

Rumpole extends the introspective reasoning to the policy decisions as well. At the policy decision level, a knowledge gap appears when the policy is underspecified for a given request. Similarly, some requests may be both explicitly permitted and denied, resulting in a conflict. Rumpole adopts an algebraic approach to specifying a break-glass policy and thus any decision gaps and conflicts are made explicit during a policy evaluation.

In summary, we see our main contributions as follows:

(1) **Rumpole** - a novel break-glass policy language for expressing policies that formulate their security decisions based on different levels of knowledge: ranging from complete gaps to strong conflicts. This is in contrast to current break-glass languages, which assume that each contextual condition can be correctly established. Rumpole adopts an algebraic approach to policy specification and thus a policy can also make its decision based on whether its own rules contain decision gaps or conflicts. We believe that this added flexibility improves the overall applicability of the break-glass approach, since this approach is precisely needed when there is a lack of knowledge about subjects and the context.
(2) **Beagle** - a novel logic-programming language for reasoning over many-valued bilattice-based truth spaces. Bilattices have been successfully applied in AI for reasoning with incomplete and incoherent knowledge. Beagle's syntax contains a

functionally-complete set of operators, which means that any truth-value operator can be captured with Beagle's core operators. In contrast, current bilattice-based logic-programming languages do not have a functionally-complete set of operators. We further define *preferred-model* semantics for a class of *stratified* Beagle programs and use it for defining Rumpole's semantics.

We provide semantics for both languages as well as detailed rationale for our design choices.

In this paper, we substantially improve our previous work on Rumpole [Marinovic et al. 2011]. First, we extend Rumpole in the number of values to represent uncertain knowledge. Second, we extend Rumpole with full range of override operators. Third, in our previous work, Rumpole's semantics was split in two disjoint parts. We rectify this through Beagle, which we use as the basis for Rumpole's unified semantics. Finally, through a use case, we demonstrate how Rumpole can formalise HIPAA-compliant break-glass policies.

The rest of this paper is structured as follows. In Section 2 we cover related work, followed by an introduction to bilattices as a means of representing gaps and uncertainties in knowledge. Section 4 presents an informal overview of the Rumpole Language in section we describe our Billatice Logic Programming Language called Beagle. Section 6 presents a more formal description of Rumpole's semantics. Section 7 is a case study on the use of Rumpole to specify overrides with respect to the Health Insurance Portability Act (HIPAA) Privacy Rule followed by a summary in Section 8.

2. RELATED WORK

Povey [Povey 2000] was amongst the first to articulate support for a break-glass concept. He argued that there is always an expressiveness gap between what can be encoded and what the needs of an organisation are. He introduced partially-formed transactions, whose effects can be *rolled-back*; a subject with no permissions could then freely execute such transactions. Risannen et al. [Rissanen et al. 2004] have similarly argued that all requests cannot be anticipated and that many conditions are not fully encodable. Their model provides the *can* predicate, which permits the requestor to override a denied decision. Medical information systems [Ferreira et al. 2009] require break-glass provisions but implement them in an ad-hoc fashion, e.g. giving *super-user* roles which have no restrictions.

Brucker et al.'s break-glass model is one of the first generic models [Brucker and Petritsch 2009]. The model views an access control policy as a partially ordered set of low-level permissions, with conditional override permissions attached to each access permission. Ardagna et al. [Ardagna et al. 2010] present another generic break-glass model, where all access control policies are split into different categories: the *standard* access control policies, *anticipated* emergency policies, and *no-restriction* break-glass policies. Unless the access is explicitly denied, it can be obtained by either finding an applicable emergency policy with obligations or, if that is not successful, the override is granted if the system is in the emergency state and the supervisor can be notified about the override. Even though these two models are generic, they still implicitly hard-code their break-glass resolution procedures, rather than, as with Rumpole, expressing it as a declarative algebraic expression. This means that a policy writer has to specify a break-policy according to a particular model's break-glass procedure, which in turn may limit the expressiveness of the intended break-glass policy. For example in Ardagna et al.'s work explicit access control denials can never be overridden, and the override depends on correctly encoding and identifying emergency situations. In contrast, our work makes no assumptions on the conditions or the context in which a break-glass policy is applied. The major difference, however, between Rumpole and the surveyed break-glass models is in the explicit reasoning over unknown and conflicting knowledge in order to permit an override. For example, in situations where it is known that the subject does not have an override permission, an override can still be given if the subject has not broken any prior obligations. Existing approaches do not attempt to recognise gaps and conflicts in a break-glass policy specification itself. They also assume that all missing information is false by default.

The problem of predicting and encoding all permissible requests has been recognised in *A posteriori* compliance control model [Etalle and Winsborough 2007] and Audit-based access control model [Cederquist et al. 2007; Hasan and Winslett 2011]. These models allow access to take place but it retrospectively checks to determine whether it conforms to policy. The difference with the break-glass approach is that the subject is unaware of any overriding and the system does not issue any obligations as a result of the overriding.

Obligations have been used to augment access control policy [Ni et al. 2008; Bettini et al. 2002; Irwin et al. 2006; Park and Sandhu 2004], in contrast to the break-glass approach obligations are issued without user interaction. A notable exceptions are languages Ponder2 [Twidle et al. 2008] and PlexC [Gall et al. 2012], whose side-effects can initiate user interaction and issue additional obligations.

[Lee et al. 2006] use policy rules with different inference strengths to facilitate conflict resolution between their conclusions. In addition, they use priority overriding between rules. This approach does not model explicit knowledge gaps and it does not propagate conflicts. Access control policy algebras, such as [Li et al. 2009; Bruns and Huth 2008; Crampton and Morisset 2012], explicitly denote conflicts and gaps in a policy specification. A policy writer can also specify further policies to resolve such issues. Rumpole's algebraic approach is inspired by Bruns and Huth's PBel [Bruns and Huth 2008] language. Rumpole can express existing policy algebras, and can furthermore express delegation operators (where one subject delegates his authority to other subjects) which the existing policy algebras do not consider.

3. PRELIMINARIES: REPRESENTING GAPS AND INCONSISTENCIES THROUGH BILATTICES

We use bilattices as underlying truth spaces for Rumpole. Their truth values represent the amount of knowledge and the amount of truth that is assigned to information used during policy evaluations. We first define bilattices and then show how the truth values of two common bilattices can be interpreted. In Section 4, we discuss a guideline for selecting bilattices.

Ginsberg [Ginsberg 1988] introduced bilattices as many-valued structures, where each truth value denotes both the amount of truth and the amount of knowledge associated with a piece of information. The amount of knowledge differentiates truth values in how much evidence (and whether it is contradictory) is present to support the associated amount of truth. To construct a many-valued structure with two orderings (i.e., amount of truth and amount of knowledge), Ginsberg combined two lattices into one many-valued structure with two orderings as follows:

Definition 3.1. [Ginsberg 1988] A bilattice is a structure $\mathcal{B} = (B, \leq_t, \leq_k, \neg)$ such that B is a non empty set containing: $(B, \leq_t)$, $(B, \leq_k)$, which are complete lattices, and $\neg$ is a unary operation on B that has the following properties: (i) if $a \leq_k b$, then $\neg a \leq_k \neg b$; (ii) if $a \leq_t b$, then $\neg a \geq_t \neg b$; (iii) $\neg\neg a = a$.

Building on top of Ginsberg's work, Fitting defined a very natural many-valued truth spaces for logic programs using the $\odot$ product:

Definition 3.2. [Fitting 1990] Let $(L, \leq)$ be a complete lattice. The structure $L \odot L(L \times L, \leq_t, \leq_k, \neg)$ is defined as follows:

— $(x_1, x_2) \leq_t (y_1, y_2)$ iff $x_1 \leq y_1$ and $y_2 \leq x_2$.
— $(x_1, x_2) \leq_k (y_1, y_2)$ iff $x_1 \leq y_1$ and $x_2 \leq y_2$.
— $\neg(x_1, x_2) = (x_2, x_1)$.

Restricting L to only $\{0, 1\}$, we obtain the smallest (non-trivial) bilattice, called $\mathcal{FOUR}$, which has four truth values. This bilattice and its logical operators correspond to Belnap logic [Belnap 1977], and is the smallest bilattice over which Rumple's semantics are defined. The four truth values in $\mathcal{FOUR}$ are: the *classical* values *t* (true) and *f* (false), and two additional ones, $\perp$, intuitively denoting lack of information (no knowledge), and $\top$, denoting inconsistency or conflict of information (*over*-knowledge).

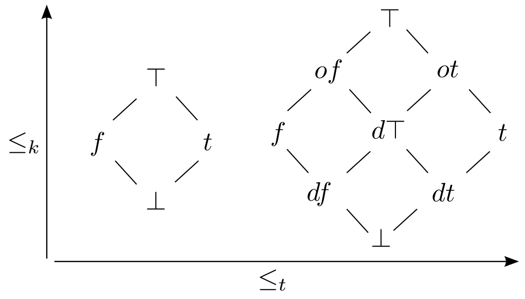

Fig. 2. The bilattices $\mathcal{FOUR}$ and $\mathcal{NINE}$ respectively.

Figure 2 orders the truth values using both the knowledge $\leq_k$, and the truth ordering $\leq_t$. The meet and join of $\leq_t$ lattice, $\wedge$ and $\vee$ respectively, correspond to the classical truth operators. Negation is represented through the unary operator $\neg$, for which $\top = \neg\top$, $\perp = \neg\perp$. The $\leq_k$, as indicated, reflects the differences in the amount of knowledge that each truth value exhibits. The four values again form a lattice such that $\top$ is the maximal element, $\perp$ is the minimal, and *t* and *f* are incomparable w.r.t. $\leq_k$. Fitting [Fitting 1991] introduced symbols $\otimes$ and $\oplus$ to denote respectively the meet and join operations. The operator $\otimes$ can be seen as giving the least amount of knowledge that the truth values can agree on, while the operator $\oplus$ can be seen as giving the most amount of knowledge that can be derived.

We adopt the following interpretation of $\mathcal{FOUR}$'s truth values:

— $\top$ – The knowledge about a piece of information is in "conflict", because it has evidence that it is both true and false.
— $\perp$ – The knowledge about a piece of information is "unknown", it has no evidence to support either claim.
— *t* – A piece of information is considered to be true, because there is evidence to support this notion and no evidence to the contrary.
— *f* – A piece of information is considered to be false, because there is evidence to support this notion and no evidence to the contrary.

$\mathcal{FOUR}$ does not represent unreliable (uncertain) knowledge levels, since an atom can only be either $true$ or $false$, in addition to gaps and conflicts. To add partial evidence, we extend L to $\{0, \frac{1}{2}, 1\}$ and obtain the bilattice $\mathcal{NINE}$ (Figure 2):

Definition 3.3. Let the lattice L be defined as $\langle\{0, \frac{1}{2}, 1\}, \leq\rangle$, then the bilattice $\mathcal{NINE}$ is defined as $L \odot L = \langle\{0, \frac{1}{2}, 1\} \times \{0, \frac{1}{2}, 1\}, \leq_t, \leq_k\rangle$, where the $\leq_k$ and $\leq_t$ are interpreted in the already described way.

We adopt the following interpretation of $\mathcal{NINE}$'s *uncertain* values:

— $dt = (\frac{1}{2}, 0)$ – The evidence to support the truth is partial or weak and there is no evidence for false.
— $df = (0, \frac{1}{2})$ – The evidence to support the falsity is partial or weak and there is no evidence for true.
— $d\top = (\frac{1}{2}, \frac{1}{2})$ – There is partial or weak evidence to support both truth and falsity.
— $ot = (1, \frac{1}{2})$ – There is strong (certain) evidence to support the truth, and weak evidence to support the falsity.
— $of = (\frac{1}{2}, 1)$ – There is strong (certain) evidence to support the falsity, and weak evidence to support the truth.

The bilattice $\mathcal{NINE}$ can be considered as a natural extension of $\mathcal{FOUR}$, because $\mathcal{NINE}$'s meet and join operators ($\wedge$, $\vee$, $\oplus$, and $\otimes$), as well as the $\neg$ operator, when restricted to Belnap values, coincide with those operators in $\mathcal{FOUR}$.

If further precision is needed to capture more subtle notions of impreciseness, we can proceed to construct $\{0, \frac{1}{3}, \frac{2}{3}, 1\} \odot \{0, \frac{1}{3}, \frac{2}{3}, 1\}$, and $\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\} \odot \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$ bilattices, ending up with 16 and 25 truth values respectively.

4. RUMPOLE LANGAUGE: AN INFORMAL OVERVIEW

In this section we present an informal overview of Rumpole. As Beagle is Rumpole's underlying formal language, we also highlight Beagle's main features. The formal presentation is in Section 5 and 6.

4.1. Policy Structure and Enforcement Model

A Rumpole policy comprises three types of rules:

(1) **Evidential Rules** – define how evidence is composed to determine how much is known about the context of an override request. For example an evidential rule may define an emergency.
(2) **Break-glass Rules** – define break-glass permissions based on override contexts. For example, a policy may say that an override is permitted in an emergency if the subject will accept an obligation to submit a reason for this override within the next four hours.
(3) **Grant Policies** – define break-glass rules are combined to reach the final override decision.

A Rumpole policy does not specify access control permissions, its aim is to encode only the break-glass permissions. Rumpole could specify some access control policies (e.g. the algebraic ones [Bruns and Huth 2008] [Crampton and Morisset 2012]) through its grant policies. We do not however pursue such a combined specification approach in this work.

Rumpole's enforcement model, depicted in Figure 3, consists of a Break-glass Policy Enforcement Point (BPEP) and a Break-glass Policy Decision Point (BPDP). The BPDP evaluates its stored Rumpole policy against override requests, which contain a subject-action-resource tuple and a set of obligations that the subject is willing to accept in order to have access to the requested object. Override requests are issued by subjects and by access control enforcement points (AC-PEPs). In the latter case, AC-PEPs act as proxies between subjects and the BPEP. An AC-PEP can be configured to automatically generate an override request for each request that it denies. The BPDP returns one of the following evaluation decisions:

(1) $grant$ – allow access to the requested resource.
(2) $request\_obligations$ – contains a set of obligations that a subject has to accept to qualify for a $grant$.

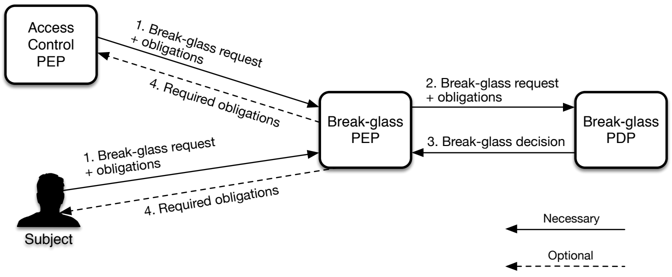

Fig. 3. Rumpole's Enforcement Model.

(3) $deny$ – deny access to the requested resource.

The BPEP enforces BPDP's decisions by allowing/denying access to requested objects for $grant/deny$ break-glass decisions. In case of a $request\_obligations$ decision, the BPEP forwards the set of required obligations that the subject has to accept. If the subject accepts the obligations, a new override request must be issued with those obligations. This is needed because the contextual conditions may have changed by the time the subject has accepted the obligations. Finally, input/output components that present requested obligations to subjects and record subjects' responses are outside of our model.

4.2. Selecting Bilattices for Rumpole Policies

The policy writer selects for each Rumpole policy one bilattice whose truth values denote the amount of knowledge and truth associated with the information used and inferred during policy evaluations. This selection depends on the trustworthiness of the policy's information sources for supplying the required evidence.

If all information sources are fully and equally trusted, then the bilattice $\mathcal{FOUR}$ should be used. In this case, evidence from all sources has the same strength in supporting the notion that some piece of information is either true or false. The gap denotes that no evidence exists, and the conflict denotes that contradictory evidence is present. To illustrate, consider a workflow break-glass policy that grants the override of a task execution if the subject has already executed the same task during the workflow's execution and accepts to perform certain additional tasks. If the task-execution databases with high integrity are the information sources, and the policy writer does not consider one database to be more trustworthy than others, then $\mathcal{FOUR}$ is sufficient.

If the policy's evidence sources are not equally trusted, then $\mathcal{NINE}$ and larger bilattices should be used. In this case, the additional truth values denote that evidence of different strength has been obtained. Intuitively, given a bilattice $L \odot L$ where L's elements are from $[0..1]$, then L's elements denote the different trust levels that can be assigned to the presented evidence that supports the notion that the information is true or false. In Section 3, we discuss such interpretations of $\mathcal{NINE}$'s truth values. To illustrate, consider a hospital break-glass policy where emergency situation information depends on evidence from both nurses and patient sensors. The policy writer may prefer the nurses' inputs over the sensors', as the sensors are known to be unreliable. The policy writer can use $\mathcal{NINE}$'s values to denote different trustworthiness of these sources. We expand this example in the rest of this section.

4.3. Evidential Rules and Beagle's Foundations

Rumpole rules are syntactically constrained Beagle rules. Evidential rules are however an exception and have no syntactic restrictions. Hence, in this subsection, we combine the overview of Beagle and evidential rules.

The Beagle language has the typical logic programming syntax:

$$p(X, a) \Leftarrow q(X) \otimes \neg r(X)$$

An atom is an expression made up of a predicate name and its arguments, e.g. $p(X, a)$, $q(X)$. We say that $p(X, a)$ is the rule's *head*, and the rule's *body* is a formula consisting of bilattice operators (Section 3) and atoms.

Atoms that represent a policy's context are referred to as *evidential* atoms. All rules that have an evidential atom as their heads are called evidential rules. Consider the following example:

$$\begin{aligned} &emergency(Patient) \Leftarrow bsnEmergency(Patient) \\ &\qquad \oplus (saysEmergency(Nurse, Patient) \wedge assigned(Nurse, Patient)) \end{aligned}$$

The body's formula combines evidence to determine the amount of knowledge assigned to the rule's head. In this case, the formula maximises the combined knowledge from the $Patient$'s body sensor network (bsn) and the $Patient$'s assigned nurse. The relation $saysEmergency(Nurse, Patient)$ defines whether a nurse has declared an emergency for a particular patient. In Beagle, the symbol $\Leftarrow$ is not the *classical* material implication, which is used in traditional logic programming. It denotes that a rule's head should have assigned to it at least the amount of knowledge as given by the evaluation of its body. Beagle's semantics, therefore, assign to each head atom only as much knowledge as can be derived from the evidence. The amount of knowledge assigned is (i) *minimal* since it uses only the evidence and the given set of rules, and (ii) *supported* meaning that all rules are taken into account.

To illustrate, consider a scenario for Patient bob monitored by a body sensor network bsn where both the *bsn* and bob's nurse $alice$ declare an emergency. We represent such a scenario as a set of input *facts* over which Beagle rules are applied:

$$bsnEmergency(bob) = t, \quad saysEmergency(alice, bob) = t, \quad assigned(alice, bob) = t$$

In this case $emergency(bob)$ will also be t, and Beagle will infer as ***true*** that bob is in an emergency. Consider a case where bob is not displaying any characteristic symptoms of an emergency, and his bsn declares the emergency as "false", but $alice$ sees some suspicious perspiration and declares an emergency giving:

$$bsnEmergency(bob) = f, \quad saysEmergency(alice, bob) = t, \quad assigned(alice, bob) = t$$

In this case $emergency(bob)$ will be $\top$. We say that policy inferences are truth values assigned to atoms based on the supplied evidences and evidential rules. When we say that Rumpole (through Beagle) infers gaps or conflicts in a policy's knowledge base, we mean that some atoms are explicitly assigned $\perp$, $\top$ or other unreliable truth values.

Beagle, and thus Rumpole, adopts the **Open-World Assumption** (OWA) for all missing facts, that is all input atoms for which no truth value is assigned, are assigned $\perp$ by default. In contrast, traditional logic programming treat missing data as $false$ by default, adopting a form of Closed-World Assumption (CWA). This can lead to erroneous conclusions. For the previous example, it would mean that a sensor being offline would be treated as $false$, i.e. equating it with no emergency.

4.3.1. Query Operator. The previously defined $emergency$ rule, though deceptively simple, can produce unexpected inferences through the $\wedge$ operator. For example, if it were the case that $alice$ is not assigned to bob and she says that there is not an emergency, this would result in the whole expression being $\top$ (given that bsn declared the emergency). This is not what we intended to capture with this rule: $alice$'s testimony should not have been taken into account. We have to limit when some evidence is *applicable* and *relevant*. To this end we introduce the *query* operator, and we apply it as follows:

$$\begin{aligned} &emergency(Patient) \Leftarrow bsnEmergency(Patient) \\ &\qquad\qquad \oplus (saysEmergency(Nurse, Patient) \otimes [assigned(Nurse, Patient) = \mathsf{t}]) \end{aligned}$$

The query operator is defined as $[\phi_1 \mathbf{\ op\ } \phi_2]$, where $\mathbf{op}$ is in $\{>_{t/k}, <_{t/k}, \leq_{t/k}, \geq_{t/k}, =\}$. In Beagle rules, constants written in the sans font represent the bilattice truth-values (e.g. t represents the truth value t, $\mathsf{d}\top$ represents $d\top$). The intuitive understanding of the query operator is as follows: return $\top$ if the comparison within the brackets holds, otherwise return $\bot$. For the example the query operator effectively blocks the nurse's evidence if she is not assigned ($assigned(Nurse, Patient) \neq t$). If the nurse was assigned, the evaluation would be $\top$ and the final result would be the truth value assigned to $saysEmergency(...)$.

The query operator is useful for defining other similar operators. For example, we take a **if** b as the syntactic shortcut for $a \otimes [b = \mathsf{t}]$. The value of a is returned only if b is evaluated as t, otherwise it is $\bot$. We further define the following syntactic shorthcut:

$$a[b \mathbf{\ op\ } c] \overset{def}{=} (a \otimes [b \mathbf{\ op\ } c])$$

This expression returns the value of a if $b \mathbf{\ op\ } c$, otherwise it is $\bot$. The query operator is $\leq_k$ non-monotonic and it cannot be captured through any combination of core bilattice operators because they are all $\leq_k$ monotonic operators. In the next section, we formally show that Beagle's set of operators is functionally-complete over its designated bilattice, so Beagle's rule body can express any operator that maps a set of input arguments to a designated resulting truth value.

4.3.2. Combining multiple rules. Complex relations between input facts and evidential atoms are often easier to capture and manage using multiple evidential rules, whose bodies are then combined for the final evaluation result. For example:

$$\begin{aligned} &bsnEmergency(Patient) \Leftarrow bsn_1(Patient) \oplus bsn_2(Patient) \\ &emergency(Patient) \Leftarrow bsnEmergency(Patient) \otimes \mathsf{d}\top \\ &emergency(Patient) \Leftarrow saysEmergency(Nurse, Patient) \textbf{ if } assinged(Nurse, Patient) \end{aligned}$$

Note that in the second rule the effect of having the $\mathsf{d}\top$ as a highest consensus value for bsn's decision is to declare the bsn as an unreliable source of evidence.

Traditional logic programming languages combine the bodies of multiple rules with the same head using the $\vee$ operator. We, however, find this to be unsuitable. Consider the following facts:

$$bsn_1(bob) = \bot,\ bsn_2(bob) = \bot,\ saysEmergency(alice, bob) = \top,\ assigned(alice, bob) = t$$

Were the rules combined in the traditional way, then $emergency(bob) = t$ would hold (since the first rule is evaluated as $\bot$ and the second as $\top$). This is not an intuitive interpretation of Beagle rules, because they combine the knowledge expressed through its rules. Beagle's semantics, therefore, uses the $\oplus$ operator to combine rule bodies. Using $\oplus$, we have $emergency(bob) = \top$. This is preferred since the rules convey how much is known in terms of knowledge ordering, rather than truth ordering. The use of $\top$ and $\oplus$ in Beagle's semantics shall be further explained in Section 5.

Existing break-glass approaches (see Section 2) cannot express Rumpole's evidential rules, and they do not consider that information may be missing and conflicting. These models are therefore not suited for break-glass applications where the system may experience failures (resulting in missing information) and use partially trusted information sources. Similarly, existing logic programming languages and many-valued policy algebras cannot be used to express and evaluate the given evidential rules. Beagle cannot be thus readily replaced as Rumpole's underlying language.

4.4. Break-glass Rules and Grant Policies

A break-glass rule is a mapping from what is known about a break-glass context and which obligations a subject has accepted into an *intermediate* break-glass decision. Intermediate decisions are later combined to make a final override decision.

Using Beagle, as is, to specify break-glass rules without any restrictions would easily lead to unintended decisions, because each rule would support both denials and permits. The reason why such issues are not present in evidential rules is that they rely on both supporting and opposing evidence to be propagated within the same rule. Rumpole's syntactic restrictions for break-glass rules are inspired by the PBel language [Bruns and Huth 2008], where a policy is an algebraic expression.

We split break-glass rules into:

(1) *Positive rules* – support granting an override, possibly given a set of obligations.
(2) *Negative rules* – support denying an override, i.e. it opposes granting an override.
(3) *Composite rules* – combine positive, negative and other composite rules.

A positive rule takes an override context and either returns t (grant an override), or $\bot$ (no decision). A negative rule, in contrast, returns f (deny an override) or $\bot$. Composite rules take positive and negative rules and combine their decisions based on various operators such as: priority, majority, and so forth. A composite rule can, furthermore, return $\top$ indicating that it was not able to resolve conflicts between its input rules. The reader will notice that we have chosen $\mathcal{FOUR}$ as the underlying bilattice for break-glass rules. This is because $\mathcal{FOUR}$ is the minimal bilattice required to indicate whether there are any gaps and inconsistencies in the rule decisions. A policy writer can expand break-glass rules so that positive and negative rules return more truth values, for example dt and t in case of positive break-glass rules. Such an extension would follow the same principles and we do not pursue it here.

4.4.1. Positive and Negative (Break-glass) rules. Positive and negative rules have the following syntax:

$$\pi(Sub, Tar, Act) \Leftarrow (\mathsf{t} \mid \mathsf{f})\ [\psi]\ \textbf{if}\ acceptedObl(...) \wedge \cdots \wedge acceptedObl(...)$$

If at least one of the required obligations was not accepted then a rule will return $\bot$. Similarly, if the conditions within $[\dots]$ are not satisfied a rule will return $\bot$. Otherwise, a rule will return t or f depending on its type. Consider the following example:

$$\pi_1(Sub, Tar, Act) \Leftarrow (\mathsf{t}[emergency(Tar) \otimes nurse(Sub) \geq_t \top])$$
$$\textbf{if}\ acceptedObl(Sub, reason, log, t_{window})$$

π_1 grants an override under the two following conditions:

(1) The expression within the query operator $[\dots]$, is satisfied – there is at least positive evidence that the request's target Tar is in an emergency and also that a nurse has requested an override.
(2) The subject Sub has accepted the proposed obligation (submitted with the request).

To illustrate how different levels of knowledge about the context can be used in Rumpole break-glass rules, we consider a scenario in which failures happen, e.g. sensors fail, responsible nurses are unavailable, etc. In these cases it may be unknown whether there is an emergency. This reasoning is *orthogonal* and *supplementary* to the choice of break-glass conditions (in this case "emergency"). Consider the policy, *an override when there is no positive evidence for an emergency is granted to nurses, but a head-nurse needs to be alerted and a nurse has to submit a reason.*

$$\begin{aligned}\pi_2(Sub, Tar, Act) \Leftarrow \mathsf{t}[&emergency(Tar) \otimes nurse(Sub) \leq_k \mathsf{d}\top]\\ &\textbf{if } acceptedObl(Sub, reason, log, t_{window})\\ &\wedge acceptedObl(sys, Tar, alert\_head\_nurse, t_{window})\end{aligned}$$

4.4.2. Composite rules and grant policies. A composite rule is a Beagle rule whose body atoms are heads of other break-glass rules. For example, a composite rules, which non-deterministically combines the given π_1 and π_2 is specified as:

$$\pi_{comp}(Sub, Tar, Act) \Leftarrow \pi_1(Sub, Tar, Act) \oplus \pi_2(Sub, Tar, Act)$$

If Sub, Tar, Act are the only variables in its body, then we usually omit them from the specification; they are be considered to be implicit. We can further compose this rule with another *conflict-resolution* rule and thus build more elaborate constructs. In Section 6, we show that due to functional-completeness of Beagle's operators over a particular bilattice, a composite break-glass rule can express any rule-combining operator. More elaborate examples of combinatorial operators are in the case study in Section 7.

Given the set of positive, negative and composite rules, we have to designate one composite rule, which will be used to derive the final override decision. We refer to this rule as a *grant policy*, and we denote it with Ω. For example:

$$\begin{aligned}&\Omega \Leftarrow (\pi_{comp} \rhd_{\perp} \pi_3)\\ &\pi_3(Sub, Tar, Act) \Leftarrow \mathsf{t}[canBeRecorded(Sub) = \mathsf{t}]\\ &\qquad\qquad\textbf{if } acceptedObl(cctv, Sub, record, t_{window})\end{aligned}$$

This grant policy evaluates π_{comp} and denies or permits if there is a conclusive decision. However, if there is a gap in π_{comp} then π_3 is consulted. The $\rhd_{\perp}$ operator is an override operator which only evaluates the right-hand policy if the left-hand policy is incomplete.

Finally, we map the evaluation of the Ω rules onto Rumpole's override decisions as:

(1) $grant$ – Ω is evaluated as t.
(2) $request\_obligations$ – There exists a set of obligations such that Ω is evaluated as t.
(3) $deny$ – Otherwise.

If conflicts and gaps are not handled then the final decision will be the most conservative one. The second decision $request\_obligations$ requires a form of a satisfiability check over the grounded grant policy expression. For example for the given Ω expression, the $request\_obligations$ is returned with the set $\{acceptedObl(sub, reason, log, t_{window})\}$ when a subject has not violated any obligations, and there is positive evidence that the patient is in emergency and sub is a nurse.

Existing break-glass approaches have fixed grant policies (see Section 2) and thus they limit the kind of policies that the policy writer can specify. In contrast, Rumpole does not use a fixed grant policy, and its algebraic approach enables it to express how different amounts of knowledge about the context are put together to reach a break-glass decision. In Section 6, we formally study Rumpole's expressiveness. Finally, we note that Rumpole can express the grant policies of all existing approaches.

5. BEAGLE: A BILATTICE LOGIC-PROGRAMMING LANGUAGE

5.1. Truth Spaces and Operators

Beagle does not have a fixed truth-space, its semantics are defined over a subset of distributive and interlaced bilattices. A Beagle program must be coupled with one bilattice from this set.

Definition 5.1. A Beagle program's underlying truth space is a bilattice $L \odot L$, where $L = \langle S, \leq \rangle$ and S a finite set of discrete numerical values in the range $[0, 1]$.

We use the term $\mathcal{B}$ to denote a bilattice that is a member of this set. The term $V_{\mathcal{B}}$ denotes the set of truth values contained in its respective bilattice. The minimal bilattice over which Beagle's semantics is defined is $\mathcal{FOUR}$. For a bilattice $\mathcal{B}$, an interpretation I over a finite set of atomic formulae is a function that assigns to every atomic formula a truth value from $\mathcal{B}$. For any formula ψ, we write $\mathcal{A}(\psi)$ for the set of atomic formulae (atoms) over which it is defined. Following Arieli and Avron [Arieli and Avron 1998], we formally define an operator as:

Definition 5.2. Given a bilattice $\mathcal{B}$, an operator g is a function $V_{\mathcal{B}}^n \mapsto V_{\mathcal{B}}$. It is represented by a formula ψ, where $\mathcal{A}(\psi) \subseteq \{p_1, \ldots, p_n\}$ (where p_i is an atomic formula), if for every interpretation I it holds: $I(\psi) = g(I(p_1), \ldots, I(p_n))$.

For the language $\mathcal{L}_{\mathcal{B}} = \{\wedge, \vee, \otimes, \oplus, \neg\} \cup V_{\mathcal{B}}$, we extend the interpretation of a formula ψ composed from its operators, constants, and atomic formulas in the usual manner. Note that this language is not *functionally-complete* over $\mathcal{B}$: it cannot express $\leq_k$ non-monotonic operators [Ruet and Fages 1997] and [Arieli and Avron 1998]. To define a functionally-complete set of operators, we first introduce the *equality* operator:

$$I(\psi \cong \phi) \overset{def}{=} \begin{cases} \top & \text{if } I(\phi) = I(\psi) \\ \bot & \text{otherwise} \end{cases}$$

With this operator, we have the following:

THEOREM 5.3. *Given a finite bilattice $\mathcal{B}$, the language $\mathcal{L}_{\mathcal{B}} \cup \cong$ is functionally complete.*

PROOF. **Case** $n = 0$: All $\mathcal{B}$'s truth values are already in $\mathcal{L}_{\mathcal{B}}^{\star}$.
Case n :

$$\psi^n \overset{def}{=} \bigoplus_{v_1 \in \mathcal{B}} \cdots \bigoplus_{v_n \in \mathcal{B}} ((p_1 \cong v_1) \otimes \ldots \otimes (p_n \cong v_n) \otimes ?g_{(v_1, \ldots, v_n)})$$

where $?g_{(v_1,\ldots,v_n)}$ stands for the evaluation $g(v_1, \ldots, v_n)$. Given ψ_n constructed in this way, and an arbitrary interpretation I of atoms $p_1, \ldots, p_n$ and an arbitrary n-tuple $(v_1, \ldots, v_n)$, then there is one and only one expression of the form $(p_1 \cong v_1) \otimes \cdots \otimes (p_n \cong v_n)$ that is evaluated as $\top$ in ψ^n. This is the expression where $v_1 = I(p_1), \ldots, v_n = I(p_n)$, while all others are evaluated as $\bot$. Thus $I(\psi^n) = \bot \oplus \cdots \oplus (\top \otimes ?g_{(v_1,\ldots,v_n)}) \oplus \cdots \oplus \bot$, where $v_1 = I(p_1), \ldots, v_n = I(p_n)$.
Since this is true for an arbitrary I and any n-tuple, it follows that
$I(\psi^n) = ?g_{(v_1,\ldots,v_n)} = g(I(p_1), \ldots, I(p_n))$. □

This theorem does not establish that the given set of operators is a minimal set needed for functional completeness.

5.2. Syntax

Beagle is a function-free logic programming language. Its rules define relationships between the degree of knowledge of truth values of a program's atomic formulae. This is in contrast to normal logic programs [Ceri et al. 1989] [Subrahmanian 1999] [Przymusinski 1988b] whose rules contribute to the degree of truth about its atomic formulas.

We assume the existence of an arbitrary but fixed first-order language $\mathcal{L}_B[\mathcal{B}]$ generated from a selected alphabet of constant symbols, variable symbols, predicate symbols, and the truth values from a designated bilattice $\mathcal{B}$. In this manner, we parametrise Beagle's syntax with a given set of truth values. A term is a variable or a constant. An atomic formula, or simply an atom, is either a truth value from $\mathcal{B}$, or an expression $p(t_1, ..., t_n)$ where p is a predicate of arity n, t_i is a term and $t_i \notin \mathcal{B}$. An atom $p(t_1, ..., t_n)$ is *ground*, if all of its terms are constants. We adopt Prolog's notation of using a capital letter to start a variable symbol and a lower case letter to start a predicate or a constant. We define an $\mathcal{L}_B[\mathcal{B}]$ formula as:

Definition 5.4. An $\mathcal{L}_B[\mathcal{B}]$ formula, ψ, is defined inductively as:

$$\psi ::= a \mid \mathsf{v} \mid \psi \oplus \psi' \mid \psi \otimes \psi' \mid \psi \cong \psi'$$

where a is an atom in $\mathcal{L}_B$, and v is a truth value in $V_{\mathcal{B}}$.

The constants in the sans font v denote the respective truth values in $\mathcal{B}$. We also use other previously introduced bilattice operators as syntactic shorthand.

Definition 5.5. A Beagle rule of language $\mathcal{L}_B[\mathcal{B}]$ is an expression of the form:

$$A \Leftarrow \psi$$

where A is an $\mathcal{L}_B[\mathcal{B}]$ atom, and ψ is an $\mathcal{L}_B[\mathcal{B}]$ formula.

A is the rule's head and ψ is the rule's body. A Beagle rule says that there is supporting evidence for a ground atom A to have at least the truth value of ψ with respect to the degree of knowledge ($\leq_k$ ordering) that the truth value has. Since variables in a Beagle rule are unbound, we extend the definition of a Beagle rule to a normalised Beagle rule as:

Definition 5.6. Let $A \Leftarrow \psi$ be a Beagle rule with variables $X_1, \ldots, X_n$ appearing in A and ψ, and variables $Y_1, \ldots, Y_m$ appearing in ψ but not in A, then its *normalised* form is defined as:

$$\forall X_1 \ldots \forall X_n (A \Leftarrow \Sigma Y_1 \ldots \Sigma Y_m\ \psi)$$

where the quantifier Σ is the infinitary (knowledge) join operation on $\leq_k$.

Intuitively, the operator Σ takes a (possibly infinite) list L of billatice truth values and returns $\perp$ if $L = \emptyset$, otherwise it returns $(x_0 \oplus x_1 \oplus \ldots)$, where x_i is an element of L and all the elements are accounted for. We bind a body's variables with the infinitary join operator Σ so that a Beagle rule accounts for all possible testimonies that can be found within a program.

Definition 5.7. A Beagle program is a finite set of normalised Beagle rules.

5.3. Supported Model Semantics

The intended domain of a Beagle program P is its Herbrand universe H_U^P, defined as a set of all ground terms that appear in P's definition. A Herbrand base H_B^P of P is the set of all ground atoms with predicate symbols from P and arguments from H_U^P.

Definition 5.8. An interpretation I_P (of a program P) is a mapping from every ground atoms in H_B^P to truth values in $\mathcal{B}$.

I_P is extended to an $\mathcal{L}_B[\mathcal{B}]$ formula as follows:

— $I_P(\mathsf{d}\top) = \top$ and similarly for other v constants.
— $I_P(\neg\psi) = \neg I_P(\psi)$ where ψ is an $\mathcal{L}_B$ formula.
— $I_P(\psi \wedge \psi') = I_P(\psi) \wedge I_P(\psi')$ where ψ and ψ' are $\mathcal{L}_B$ formulae, and similarly for other operators.
— $I_P(\forall X \psi(X)) = \bigwedge_{x \in H_U^P} I_P(\psi(x))$.
— $I_P(A \Leftarrow \psi) = \begin{cases} t & \textit{if } I_P(A) \geq_k I_P(\psi) \\ f & \textit{otherwise} \end{cases}$
— $I_P(\Sigma X \psi(X)) = \bigoplus_{x \in H_U^P} I_P(\psi(x))$.

We define a *model* of P as:

Definition 5.9. An interpretation of P is a model of P, denoted as M_P, iff for every rule $\mathcal{B}$ in P it holds: $M_P(\mathcal{B}) = t$.

This definition emphasises that a rule $a \Leftarrow \psi$ is interpreted as $M_P(a) \geq_k I(\psi)$.
For a Beagle rule $\mathcal{B}$, $A \Leftarrow \psi$, we associate a set $ground(\mathcal{B})$ containing the ground rules constructed through the following two steps:

(1) Consistently substitute ground terms from H_P^U for each variable in $\mathcal{B}$'s head to produce a set of rules $a \Leftarrow \Sigma Y_1 \ldots \Sigma Y_m\ \psi'$, where a is a ground term.
(2) For each rule from (1), consistently substitute ground terms from H_P^U for each variable in ψ' to produce $a \Leftarrow \psi_1 \oplus \cdots \oplus \psi_k$.

For a Beagle program P we associate a ground Beagle program $P^\star$ such that: $P^\star = \bigcup_{\mathcal{B} \in P} ground(\mathcal{B})$. The following theorem establishes that P and $P^\star$ have exactly the same models:

THEOREM 5.10. *An interpretation I is a model of P iff it is a model of $P^\star$.*

PROOF. We can obtain a proof by contradiction in both directions:
Take I as a model of P then there is one arbitrary rule $\mathcal{B}^\star$ (with some a as head) in $P^\star$ for which $I(\mathcal{B}^\star) = f$. There must be a rule $\mathcal{B}$ of which the rule $\mathcal{B}^\star$ is a ground instance. From the grounding of $\mathcal{B}^\star$ it follows that $I(\psi_1 \oplus \cdots \oplus \psi_k) = I(\bigoplus Y_1 \cdots \bigoplus Y_m\ \psi)$, since a free variable is uniformly replaced with each constant (as described in step 2 of the grounding). It follows that $I(a) \not\geq_k I(\bigoplus Y_1 \cdots \bigoplus Y_m\ \psi)$, which implies that $I(\mathcal{B}) = f$, which is a contradiction since I is a model of P.
Take I to be a model of $P^\star$ then assume that there is one arbitrary rule $\mathcal{B}$ in P for which $I(\mathcal{B}) = f$. It follows that there must exist at least one partially ground instance of $\mathcal{B}$ for which $I(a) \not\geq_k I(\bigoplus Y_1 \cdots \bigoplus Y_m\ \psi)$. By its construction, $ground(\mathcal{B})$ will contain a rule $a \Leftarrow \psi_1 \oplus \cdots \oplus \psi_k$ which will be a fully grounded instance $\mathcal{B}^\star$ of $a \Leftarrow \bigoplus Y_1 \cdots \bigoplus Y_m\ \psi$. It follows that $I(\mathcal{B}^\star) = f$, but this is a contradiction since I is a model of $P^\star$. □

P's model can *over*-assign knowledge, for example by assigning $\top$ to some atoms without any rule's support for such a value. To ensure that only the program's rules influence a model, we define a *supported* model as:

Definition 5.11. A model M_P of P is a supported model if for every $a \in H_B^P$ it holds:

$$M_P(a) = M_P(\psi_1) \oplus \cdots \oplus M_P(\psi_n)$$

where ψ_i is a body of a rule $a \Leftarrow \psi_i$ in P.

LEMMA 5.12. *I is a supported model of P iff I is a supported model of $P^\star$.*

PROOF. By contradiction in both directions. □

A supported model uses $\oplus$ to enforce the intended reading of a Beagle program as a join of all the given testimonies. Consider the following P:

$$p \Leftarrow w \vee r \quad p \Leftarrow q \quad w \Leftarrow \mathsf{t} \quad r \Leftarrow \bot \quad q \Leftarrow \mathsf{f}$$

A supported model M_P assigns $\top$ to p. In standard logic programs, the bodies are combined using the $\vee$ operator. Had we adopted this approach, $I_P(\psi_i) \vee I_P(\psi_{i+1})$, we would have $M_P(p) = t$. This assignment would ignore the reliable testimony that q is at least f, and thus that p should be at least f as well. If P is changed by modifying $q \Leftarrow \top$ and $w \Leftarrow \bot$, then an $\oplus$-based supported model has $M_P(p) = \top$, but the $\vee$-based model results in $M_P(p) = t$. This discussion also clarifies the reason for binding a body's free variables with Σ.

A supported model, however, is not guaranteed to exist. The program: $\{p \Leftarrow (p \cong \top_{\mathcal{B}}) \cong \bot_{\mathcal{B}}\}$ has one model $\{p = \top_{\mathcal{B}}\}$, which is not supported. A Beagle program can also have many supported models, just as is the case with normal logic programs.

In the rest of this section, we impose syntactic restrictions on a Beagle program to ensure that supported models exist, and then we select one *canonical* model to represent its *preferred model* semantics. The same approach is applied to stratified semantics in normal logic programs [Przymusinski 1988b].

5.4. Minimal Supported OWA Model

We can compare supported models is in the amount of knowledge that they assign to individual atoms.

Definition 5.13. For models M_P and M'_P of a program P it holds $M'_P \leq_k M_P$ if for every ground atom a it holds $M'_P(a) \leq_k M_P(a)$.

We say that $M'_P <_k M_P$ if $M'_P(a) \leq_k M_P(a)$ and there is at least one ground atom a such that $M'_P(a) <_k M_P(a)$. We define a *minimal* supported model as:

Definition 5.14. A supported model M_P of a program P is minimal if there does not exist another supported model M'_P of P, such that $M'_P <_k M_P$.

LEMMA 5.15. *A model M is a minimal supported model of P iff it is a minimal supported model of $P^\star$.*

PROOF. By contradiction in both directions. □

Before constructing a minimal supported model, we need to address how missing facts are treated in Beagle programs. Consider the following program:

$$p \Leftarrow q \wedge w \qquad w \Leftarrow \neg r$$

There are no rules to infer the truth values for atoms r and q. Normal logic programs use a form of Closed-World Assumption (CWA) [Przymusinski 1988a] to assume that all facts that are not specified are by default f. In case of Beagle programs, it is appropriate to adopt the Open-World Assumption [Reiter 1977] (OWA), whereby the missing information is assigned the $\bot$ truth value, because it fits our intuitive understanding of $\bot$ as the "unknown" truth value. Formally:

Definition 5.16. An interpretation I_P of a Beagle program P is an OWA model of P, denoted M_P^{OWA}, if it is a model of P and for every ground atom a, which is not in a head of any rule in P, it holds that $M_P^{OWA}(a) = \bot$.

Fitting introduced a mapping operator T_P [Fitting 1990] that maps an interpretation I of a ground logic program $P^\star$, whose truth values are based on an interlaced bilattice, onto an interpretation I', also of $P^\star$, in the following way:

$$T_P(I)(a) = \begin{cases} I(\psi_1 \oplus ... \oplus \psi_n) & \forall a \ where \ a \Leftarrow \psi_i \in P^\star \\ \bot & \nexists a \ where \ a \Leftarrow \psi_i \in P^\star \end{cases}$$

If only $\leq_k$ monotonic operators appear in rules, then T_P is a *monotonic* operator. Thus for any two interpretations it holds that: $I_1 \leq_k I_2 \Rightarrow T_P(I_1) \leq_k T_P(I_2)$. Fitting showed that the operator T_P is also *(chain) continuous* (for any chain of interpretations $I_1 \leq_k I_2 \leq ...$ it holds that: $T_P(\Sigma_i I_i) = \Sigma_i T_P(I_i)$). Based on the Knaster-Tarski theorem [Tarski 1955], a monotonic and continuous operator T_P has the least fixpoint $T_P{\uparrow}^\omega$, given as:

$$I_P^0 \overset{def}{=} I_\bot$$
$$I_P^{n+1} \overset{def}{=} T_P(I_P^n)$$
$$\cdots$$
$$T_P{\uparrow}^\omega \overset{def}{=} \Sigma_{\alpha<\omega} I_P^\alpha$$

where Σ is the infinitary join operator on $\leq_k$ ordering and $I_\bot$ assigns $\bot$ to every atom from P's Herbrand base.

THEOREM 5.17. *Given a ground Beagle program $P^\star$ with only $\leq_k$ monotonic operators, then its least fixpoint, $T_P{\uparrow}^\omega$, is its unique minimal supported OWA model.*

PROOF. First, $T_P{\uparrow}^\omega$ is a supported OWA model. Second, $T_P{\uparrow}^\omega$ is the *least* fixpoint, so it follows that $T_P{\uparrow}^\omega$ is the *unique* minimal supported OWA model of $P^\star$. □

5.5. Iterated Fixpoint OWA Model

A Beagle program with a $\leq_k$ non-monotonic operator is not guaranteed to have the least fixpoint. Similarly there can be more than one minimal supported model. Having $\leq_k$ non-monotonic operators, however, greatly increases the expressivity of Beagle rules. In the following, we introduce a syntactic restriction called stratification, which guarantees the existence of a minimal supported OWA model and provides an intuitive method to select a *preferred* model when many exist. This follows the same reasoning underpinning locally stratified semantics for normal logic programs [Przymusinski 1988b].

Stratification splits a program into a set of strata. Each stratum defines evaluations for a subset of the program's Herbrand base, and only monotonic operators can be used in recursive relations within a stratum. A higher stratum can query a lower stratum about the values that it assigns to atoms in the heads of the lower stratum's rules. To this end, we introduce a generalisation of the $\cong$ operator, called the *query* operator, defined as:

$$I([\phi_1 \ \mathbf{op} \ \phi_2]) \overset{def}{=} \begin{cases} \top & \text{if } I(\phi_1) \ \mathbf{op} \ I(\phi_2) \\ \bot & \text{otherwise} \end{cases}$$

where $\mathbf{op} \in \{=, \neq, <_{t/k}, >_{t/k}, \leq_{t/k}, \geq_{t/k}\}$. I

Using the *query* operator, we slightly redefine the set of core operators as:

Definition 5.18. An $L_B[\mathcal{B}]$ formula in a stratified Beagle program contains the operators: $\{\wedge, \vee, \neg, \otimes, \oplus, \textit{query}\} \cup V_{\mathcal{B}}$.

We define a *top* query formula of a Beagle rule as:

Definition 5.19. Given a Beagle rule $A \Leftarrow \psi$, then its $\top_Q$ is defined as a set of all ψ's atoms that appear in its query operators and their sub-formulae.

To illustrate, consider $A \Leftarrow a[(b \oplus c) = \mathsf{t}] \otimes d[g = \mathsf{f}]$, then its $\top_Q = \{b, c, g, f\}$. Now, we define a stratified Beagle program as:

Definition 5.20. A ground Beagle program $P^\star$ is stratified in n strata, $P^\star = P_1^\star \cup \ldots \cup P_n^\star$ if for every atom p (in $H_B^{P^\star}$):

(1) The definition of p (all rules with p as their heads) is in one stratum.
(2) If p depends on q, then q's definition is in a stratum j, where $j \leq stratum(p)$.
(3) If q is in the $\top_Q$ of a rule with head p, then q's definition is in a stratum j, where $j < stratum(p)$.

LEMMA 5.21. *Given a stratified Beagle program P, if there is a cyclic dependency between two ground atoms, then those two atoms must be in the same stratum.*

PROOF. The proof follows from the last two conditions of Definition 5.20. □

A stratified Beagle program is an ordered list of programs, P_i, where i is a particular stratum's index. Every P_i depends on the "lower" P_js ($j < i$) to determine the truth values of atoms appearing within its query formulas.

To formalise this intuition we introduce the $\sqcup$ operator, which *merges* the evalutions of two interpretation as follows. Given I_1 and I_2 with their respective (possibly overlapping) domains D_1 and D_2, then $I_1 \sqcup I_2$ is an interpretation that assings to a the value of $I_1(a) \oplus I_2(a)$ if a is in both domains, otherwise it assigns the value given by the interpretation that has a in its domain.

The stratified semantics are defined with an *iterated fixpoint* model as follows:

Definition 5.22. For a ground stratified program, $P^\star = P_1^\star \cdots \cup P_n^\star$, an iterated fixpoint model $\mathit{IFM}_{P^\star}$ is defined as:

$$\begin{aligned} I_1 &= T_{P_1^\star}\uparrow^\omega \\ I_2 &= T_{P_2^\star \sqcup I_1}\uparrow^\omega \\ &\ldots \\ I_n &= T_{P_n^\star \sqcup I_{n-1}}\uparrow^\omega \end{aligned}$$

where $T_{P_i^\star \sqcup I_{i-1}}\uparrow^\omega$ is defined as:

$$\begin{aligned} I^0_{P_i^\star \sqcup I_{i-1}} &\overset{def}{=} I_{i_\perp} \sqcup I_{i-1} \\ I^{n+1}_{P_i^\star \sqcup I_{i-1}} &\overset{def}{=} T_{P_i^\star}(I^n_{P_i^\star \sqcup I_{i-1}}) \sqcup I_{i-1} \\ &\ldots \\ T_{P_i{}^\star \sqcup I_{i-1}}\uparrow^\omega &\overset{def}{=} (\Sigma_{\alpha<\omega} I^\alpha_{P_i^\star \sqcup I_{i-1}}) \sqcup I_{i-1} \end{aligned}$$

This definition adds the minimal supported model of the lower stratum (I_{i-1}) as the input for the initial minimal interpretation $I_{i_\perp}$ of the stratum's atoms for which we are about to construct the fixpoint model (e.g. $P_i^\star \sqcup I_{i-1}$). Because the stratum i does not have any rules to regenerate I_{i-1} we have to propagate this interpretation in each application of T_P.

Note that T_P is still not monotonic in general (due to the presence of query operators). However, since the given construction fixes the values of all atoms defined in lower strata, then they can be replaced with the truth constants in the given input. In this way, the application of T_P for a stratum does not exhibit non-monotonic behaviour. Furthermore, each application of the T_P for a stratum produces a unique minimal supported model for the stratum rules when the atoms defined in the lower strata are replaced with the corresponding values from the given input. Finally, the fixpoint of the last stratum is taken as the canonical model of $P^{\star}$.

Given the previous definition, we can state the following theorem:

THEOREM 5.23. *An iterated fixpoint model* $\textit{\textbf{IFM}}_{P^{\star}}$ *of a ground stratified Beagle program* $P^{\star}$ *is also its minimal supported OWA model.*

PROOF. We can see that $\textit{\textbf{IFM}}_{P^{\star}}$ is a supported OWA model. Now, assume that $\textit{\textbf{IFM}}_{P^{\star}}$ is not a minimal supported model. This implies that there exists another supported model, M, such that $M <_k \textit{\textbf{IFM}}_{P^{\star}}$. Therefore there is an atom p such that $M(p) <_k \textit{\textbf{IFM}}_{P^{\star}}(p)$.

If p depends only on the atoms from the same stratum it is not possible to have a lower value of p and still have a supported model, since $\textit{\textbf{IFM}}_{P^{\star}}$ contains the unique minimal supported model for that strata. Therefore, ψ must depend on an atom q from a lower stratum, such that it also holds that $M(q) <_k \textit{\textbf{IFM}}_{P^{\star}}(q)$ otherwise $M <_k \textit{\textbf{IFM}}_{P^{\star}}$ would not hold.

Inductively the same analysis is applied to q, and again we are forced to find some atom r from a lower stratum. Finally, we are forced to find an atom from $I_{\perp}$ and these values are already minimal. Therefore in order to decrease a value of some p and still keep a supported model we are forced to increase a truth value for some q from a lower stratum on which p is directly or indirectly dependent. Hence this results in $M \not<_k \textit{\textbf{IFM}}_{P^{\star}}$. It follows that $\textit{\textbf{IFM}}_{P^{\star}}$ is a minimal supported OWA model. □

Since $\textit{\textbf{IFM}}_{P^{\star}}$ is a minimal supported OWA model, it follows our original intuition underpinning the semantics of Beagle programs. We stress, however, that that this result *does not* establish that $\textit{\textbf{IFM}}_{P^{\star}}$ is a unique minimal supported OWA model. Consider the following example:

$$a \Leftarrow \mathsf{t}\ \textbf{if}\ b \oplus \mathsf{t} \qquad b \Leftarrow \neg b$$

In this case, there are two minimal supported OWA models, namely $\{b = \perp, a = t\}$ and $\{b = df, a = \perp\}$. The former one is the canonical model chosen by the stratified semantics. We argue that this model is the intuitive one because a lower layer ought to contain as little knowledge as possible.

The following theorem establishes that the semantics are independent of the exact stratification:

THEOREM 5.24. *Given a stratified Beagle program* $P^{\star}$*, the iterated fixpoint model* $\textit{\textbf{IFM}}_{P^{\star}}$ *constructed over one particular stratification is the same iterated fixpoint model obtained over any other stratification of* $P^{\star}$*.*

PROOF. The proof is obtained by considering whether the differences between two stratifications of $P^{\star}$, $Q_1 \cup \cdots \cup Q_n$ and $R_1 \cup \cdots \cup R_m$, can yield different results, for an arbitrary atom p, through their respective iterative fixpoint models Q_n and R_m.

First, the definition of an atom cannot be split between different strata. All of its rules must be confined in the same stratum.

Second, if p's definition depends on q's, but q's definition also depends on p's, then by Lemma 5.21 q's rules are also in p's stratum in both stratifications.

Third, if p depends on some q through a query operator, q's definition is in a lower stratum for both stratifications. Note, that in precisely which lower stratum is irrelevant since once calculated all values are continuously propagated to all upper strata.

Thus, the only way stratifications can differ for an arbitrary p is in placing an atom q, on which p depends, in a different lower stratum, unless p is cyclically dependent on q. We do not have to concern ourselves with atoms that p does not depend on, since they do not influence p's calculated value.

Note that $T_{Q_n \cup I_{n-1}}$ and $T_{R_m \cup I_{m-1}}$ cannot compute different values for p if q's value is pre-computed in I_{n-1}, or if q's value is computed together with p's in the same stratum (Q_n or R_m). The number of steps taken by $T_{Q_n \cup I_{n-1}}$ is different, but not its final result. The same argument is made for any other dependency of p.

The same argument is inductively extended to each of p's dependent atoms, until the lowest stratum (containing only facts) which is the same for any two stratifications. Therefore calculating p's value does not depend on the placement of its dependent atoms' definitions in a particular stratum, assuming it obeys the stratification requirements. □

Finally, for a stratified Beagle program P, we take the its $\textit{\textbf{IFM}}_{P^\star}$ as its designated denotational semantics.

5.6. Beagle's Data Complexity

The time complexity of constructing a canonical model is often referred to as *data* complexity in the logic programming literature [Schlipf 1995] [Dantsin et al. 2001]. The complexity is measured with respect to the size of the facts (atoms that are not a head of any clause). We similarly define the data complexity for a stratified Beagle program $P^\star$ as the computational complexity of constructing $\textit{\textbf{IFM}}_{P^\star}$ measured as a function of the size of the $P^\star$'s facts. We refer to a particular fact set as F, and $|F|$ denotes the size of F, i.e. the total number of (constant) symbols needed to write F. Similarly to normal logic programs, we observe that the grounding of P into $P^\star$ has a polynomial time computational complexity with respect to P's $|F|$.

THEOREM 5.25. *Given a ground Beagle program, $P^\star$, containing only $\leq_k$-monotonic operators, its data complexity is polynomial time.*

PROOF. The proof follows the similar proof given for well founded semantics [Gelder et al. 1991].

Let $n = |F|$. Let a be the maximum arity of any predicate of $P^\star$. For each truth value, v, from $\mathcal{B}$ we define a set, S_v, that holds an atom if and only if T_P assigns it value t such that $v \leq_k t$.

Rather than implementing T_P for stratified semantics, we take its *progressive* version, T'_P defined as $T'_P(I) = I \oplus T_P(I)$. We can see that T'_P is monotonic, and based on Fitting's distributivity laws for bilattices, we also have that it is continuous as well:

$$T'_P(\Sigma_i I_i) = T_P(\Sigma_i I_i) \oplus \Sigma_i I_i = \Sigma_i T_P(I_i) \oplus \Sigma_i I_i = \Sigma_i (T_P(I_i) \oplus I_i) = \Sigma_i T'_P(I_i)$$

Now we show that T_P and T'_P have the same minimal fixpoint:

$$T'_P{\uparrow}^0 = I^\perp = T_P{\uparrow}^0$$
$$T'_P{\uparrow}^1 = T_P{\uparrow}^1$$
$$T'_P{\uparrow}^2 = T'_P(T'_P{\uparrow}^1) = T_P(T'_P{\uparrow}^1) \oplus T'_P{\uparrow}^1 = T_P(T_P{\uparrow}^1) \oplus T_P{\uparrow}^1 = T_P{\uparrow}^2 \oplus T_P{\uparrow}^1$$

Assuming that $T'_P{\uparrow}^n = \Sigma_i^n T_P{\uparrow}^i$, we have:

$$T'_P{\uparrow}^{n+1} = T'_P(T'_P{\uparrow}^n) = T_P(T'_P{\uparrow}^n) \oplus T'_P{\uparrow}^n =$$
$$T_P(\Sigma_i^n T_P{\uparrow}^i) \oplus \Sigma_i^n T_P{\uparrow}^i = \Sigma_i^n T_P(T_P{\uparrow}^i) \oplus \Sigma_i^n T_P{\uparrow}^i = \Sigma_i^{n+1} T_P{\uparrow}^i$$

The number of times T'_P is applied for each atom is at most the number of truth values of $\mathcal{B}$. This is because at every step T'_P adds knowledge and it settles monotonically on one value. Therefore the greatest number of steps that T'_P can be applied is the combined size of all S_v sets. The size of each set is bound by the size of $P^\star$'s Herbrand base, which in turn is polynomial to n because its size is given as $n^a + \cdots + n^0$. Therefore constructing $T'_{P^\star}{\uparrow}^\omega$ is polynomial in $|F|$. □

The corollary of the previous theorem is that the iterated fixpoint construction is also polynomial.

6. RUMPOLE: A FORMAL ACCOUNT

We briefly recall the structure of Rumpole's override policy:
(1) Evidential Rules – define how evidence facts determine how much is known about the context surrounding the break-glass request.
(2) Break-glass Rules – map a context of the override request including all accepted obligations into intermediate break-glass decisions.
(3) Grant Policies – combine all intermediate break-glass decision into a final override decision.

6.1. Rumpole Break-glass Policy Specification

We define a rumpole break-glass policy as a mapping from a set of requests, accepted obligations, and the context of a request to the override decisions:

Definition 6.1. A Rumpole break-glass policy Π is a mapping:

$$Req \times AcceptedObls \times Context \mapsto \mathcal{D}.$$

— Req is the set of override requests, $\langle sub, tar, act \rangle$ tuples, over which Π is defined.
— $AcceptedObls$ is the set of obligations that a subject has accepted.
— $Context$ is the set of all evidential atoms.
— $\mathcal{D} = \{grant, deny, request\_obls\}$

To specify and evaluate a break-glass policy Π, we use Beagle rules and Beagle's stratified semantics. The predicates, which appear in a Rumpole policy, are split into the following disjoint sets: $\mathcal{L}^{Aux}$, $\mathcal{L}^{Obl}$, $\mathcal{L}^{BG}$, and $\mathcal{L}^{Grant}$. The set $\mathcal{L}^{Aux}$ is an open set and contains all the auxiliary predicates that describe the context of a break-glass request. The set $\mathcal{L}^{Obl}$ is closed and contains only one predicate: $acceptedObl$ $(Subject \times Target \times Action \times Time)$, which denotes whether the subject has accepted to perform the requested obligatory action on the designated target within the time window designated by the last argument. The set $\mathcal{L}^{BG}$ is open and contains only the following type of predicates: π_i $(Subject \times Target \times Action)$, where $i \in \mathbb{N}_0$. The set $\mathcal{L}^{Grant}$ is closed and contains: Ω $(Subject \times Target \times Action)$. Formally:

Definition 6.2. A Rumpole break-glass policy Π predicate set $\mathcal{L}_\Pi$ is defined as:

$$\mathcal{L}_\Pi = \mathcal{L}_\Pi^{Aux} \cup \mathcal{L}_\Pi^{BG} \cup \mathcal{L}^{Obl} \cup \mathcal{L}^{Grant}$$

where $\mathcal{L}_\Pi^{Aux} \supseteq \mathcal{L}^{Aux}$.

We also define Π's context state as follows:

Definition 6.3. Π's context state, $\mathcal{C}_\Pi$, is a set of all ground atoms with predicates from $\mathcal{L}_\Pi^{Aux} \cup \mathcal{L}_\Pi^{Obl}$ grounded over constants from Π's Herbrand universe.

An interpretation of $\mathcal{C}_\Pi$ is denoted as $I(\mathcal{C}_\Pi)$.

Given these definitions, we formally define a Beagle-based specification of a Rumpole policy as:

Definition 6.4. A Rumpole policy Π is a set of Beagle rules Δ_Π such that $\Delta_\Pi = \Delta_\Pi^{Aux} \cup \Delta_\Pi^{BG} \cup \Delta_\Pi^{Facts}$, where:

— Δ_Π^{Aux}'s rules have only predicates from $\mathcal{L}_\Pi^{Aux}$ as heads.
— Δ_Π^{BG}'s rules have only predicates from $\mathcal{L}_\Pi^{BG} \cup \mathcal{L}^{Grant}$ as heads.

— Δ_{Π}^{Facts}'s rules have only $\mathcal{L}_{\Pi}^{Aux}$'s atoms as heads and their bodies are truth-value constants. No head in Δ_{Π}^{Facts} is a head in Δ_{Π}^{Aux}.
— Δ_{Π} forms a stratified Beagle program.

We denote Π's knowledge base with Δ_{Π}. The set Δ_{Π}^{Facts} represents collected facts that a Rumpole implementation (Policy Decision Point) would collect before making an override decision. Δ_{Π}^{Aux} are evidential rules and Δ_{Π}^{BG} are break-glass rules.

6.2. Override Request and Decision

An override request consists of a mandatory triple $\langle sub, tar, act\rangle$ and an optional set of obligations that a user is willing to accept. $\Delta_{AcceptedObl}$ represents which obligations have been accepted by which subjects. If $\Delta_{AcceptedObl}$ is empty, then all ground $acceptedObl$ atoms will be $\bot$.

To evaluate Π against a request $\langle sub, tar, act\rangle$, a PDP takes $\Delta_{\Pi} \cup \Delta_{AcceptedObl}$ and constructs a minimal supported OWA model denoted as $\boldsymbol{IFM}_{\Pi}$:

Definition 6.5. Given Π's specification Δ_{Π} and its designated bilattice $\mathcal{B}_{\Pi}$, a PDP grants an override for a request $\langle sub, tar, act\rangle$ and its $\Delta_{AcceptedObl}$ iff

$$\boldsymbol{IFM}_{\Pi} \models \Omega(sub, tar, act) = t$$

where: $\Pi = \Delta_{\Pi} \cup \Delta_{AcceptedObl}$.

If a subject is denied an override, a PDP evaluates the policy specification to find all combinations of obligations which, if accepted, would result in the override grant. Formally, the $request\_obligations(\mathcal{A})$ is defined as a satisfiability relation:

Definition 6.6. Rumpole PDP returns $request\_obls(\mathcal{A})$ decision for the request (sub, tar, act) and the policy specification Δ_{Π} iff

$$\boldsymbol{IFM}_{\Delta_{\Pi} \cup I^{t}(\mathcal{A})} \models \Omega(sub, tar, act) = t)$$

where $I^{t}(\mathcal{A})$ assigns t to all $a \in \mathcal{A}$. All the sets that are found are presented to the subject for possible acceptance. If obligations are accepted, the override is not automatically granted but a new override request is formed with the same sub, tar, act constants and $\Delta_{AcceptedObl} = \mathcal{A}$.

6.3. Specifying Evidential Rules

Evidential rules map fact atoms onto evidential atoms, they can also further combine and map evidence onto other evidential atoms. First, we define Π's set of evidential atoms as:

Definition 6.7. Given a policy Π, Σ_{Π}^{Aux} is a set of all ground atom with predicates from $\mathcal{L}_{\Pi}^{Aux}$.

We say that a Π's evidential atom is a ground atom from Σ_{Π}^{Aux}.

Given an override policy Π's predicate set, an evidential rule is defined as:

Definition 6.8. An evidential rule is a Beagle rule containing only $\mathcal{L}_{\Pi}^{Aux}$ predicates.

We do not allow break-glass decisions $acceptedObl$ atoms to appear in an evidentail rule because we want to enforce a separation of concerns between different rules. In particular, evidential rules should only describe the context of a request, and break-glass rules are the ones to combine this evidence to reach a policy decision.

If a particular evidential atom s depends only on facts (put into Δ_{Π}^{Facts}), then we can state the following theorem, which establishes that any such mapping from facts onto evidence can be expressed in Rumpole:

THEOREM 6.9. *Let I be an interpretation of the set of Π's ground facts, Σ_{Π}^{Facts}. Then for a given ground evidential atom s and a given function $g : \mathcal{B}_{\Pi}^{n} \mapsto \mathcal{B}_{\Pi}$ there is a Beagle rule, $s \Leftarrow \psi$, such that:*

$$\textbf{\textit{IFM}}_{\Pi}(s) = \textbf{\textit{IFM}}_{\Pi}(\psi) = g(\textbf{\textit{IFM}}_{\Pi}(p_1), \dots, \textbf{\textit{IFM}}_{\Pi}(p_n))$$

where $p_i \in \Sigma_{\Pi}^{Facts}$.

PROOF. First observe that Beagle contains a functionally-complete set of operators. Following Theorem 5.3, for any operator $g(X_1, \dots, X_n) : \mathcal{B}_{\Pi}^{n} \mapsto \mathcal{B}_{\Pi}$, there exists a Beagle formula ψ, such that $I(\psi(p_1, \dots, p_n)) = g(I(p_1), \dots, I(p_n))$, where I is an arbitrary interpretation of p_i's.

Since $\textbf{\textit{IFM}}_{\Pi}$ is a minimal supported model of the Beagle program containing $s \Leftarrow \psi$, and s is not the head of any other rule, and p_i is a ground fact atom, it follows that $\textbf{\textit{IFM}}_{\Pi}(s) = \textbf{\textit{IFM}}_{\Pi}(\psi)$. Then replacing p_is with Π's facts gives the proof for the theorem, since no other rule with s as head is added to Δ_{Π}^{Aux}. $\square$

The corollary of this theorem is that any non-recursive mapping between evidential atoms can be expressed as long as it is not recursive.

6.4. Specifying Break-glass Rules

Rumpole break-glass rules are split into three types: (1) Positive rules, (2) Negative rules, and (3) Composite rules. Positive rules can only grant an override, while negative rules can only deny an override. Composite rules explicitly combine *low-level* positive and negative rules to express more complex policies that may include conflict resolution strategies, votings, overrides and so forth.

Formally, positive and negative rules are defined as:

Definition 6.10. Let $\mathcal{C}_{\Pi}$ be a context state, $\mathcal{I}(\mathcal{C}_{\Pi})$ a set of all possible interprations of $\mathcal{C}_{\Pi}$, then a positive/negative break-glass rule π_i is a set of mappings $\pi_{i_{\langle sub,tar,act \rangle}} : \mathcal{I}(\mathcal{C}_{\Pi}) \mapsto \{grant/deny, gap\}$, for each override request $\langle sub, tar, act \rangle$.

Composite rules are defined as:

Definition 6.11. A composite break-glass rule, π_i, is a set of mappings $\pi_{i_{\langle sub,tar,act \rangle}} : \mathcal{I}(BG_{\Pi}) \mapsto \mathcal{FOUR}$, where $\mathcal{I}(BG_{\Pi})$ is the set of all possible interpretations of Π's break-glass rules.

We insist that the composite break-glass rules use four policy decisions $\{grant, deny, gap, conflict\}$ (denoted as t, f, $\bot$, and $\top$ respectively). This interpretation follows PBel's usage of bilattice $\mathcal{FOUR}$ to allow policy rules to propagate more information than just grant or deny. This is useful when a policy rule cannot make a decision (for example it is not applicable), or it itself does not want to resolve a conflict. We support the argument, made by PBel, that this four valued approach is more suitable for complex policy compositions. Each break-glass rule is specified as one or more Beagle rules with π_i as their heads as follows:

Definition 6.12. A positive/negative break-glass rule π_i is specified with Beagle rules:

$$\pi_i(Sub, Tar, Act) \Leftarrow ([\mathsf{t}|\mathsf{f}]\ [\psi])\ [\textbf{if}\ \phi_{Obl}(Sub, Tar, Act)]$$

— $\pi_i \in \mathcal{L}_{\Pi}^{BG}$.
— ψ is a Beagle formula containing only predicates from $\mathcal{L}_{\Pi}^{Aux}$.
— $\phi_{Obl}(Sub, Tar, Act)$ is a Beagle formula with predicate symbols from $\mathcal{L}^{Obl}$ and $\wedge$.

This definition insists that free variables cannot appear in the obligation formula, because free variables would be implicitly bound and potentially a large number of obligations would end up being issued.

Definition 6.13. A composite break-glass rule π_i is specified with Beagle rules:

$$\pi_i(Sub, Tar, Act) \Leftarrow \psi$$

— $\pi_i \in \mathcal{L}_{\Pi}^{BG}$.
— ψ is a Beagle formula containing only predicates from $\mathcal{L}_{\Pi}^{BG}$.

Composite rules do not have any explicit obligations attached to them. They are contained within the positive and negative rules to make the composition of rules easier to specify and manage.

Finally, we insist that there are no cyclic dependencies between break-glass rule atoms in Δ_{Π}^{BG}. If there were cyclic dependencies, break-glass rules could be defined only in terms of themselves without considering obligations or evidence:

Definition 6.14. Δ_{Π}^{BG} is cyclic-free if there are no cyclic dependency between ground $\mathcal{L}_{\Pi}^{BG}$ atoms.

This definition does not exclude acyclic dependencies, which allow a policy writer to express a closure operator whereby a policy decision for one request depends on a policy decision for another request. For example, a subject cannot have read access to both the record a and the record b. From the outset, this is a cyclic relationship, and explicitly expressing it as such is not syntactically allowed. However, using composite break-glass rules, the policy writer can break such cyclic relationships and capture the closure operator, as demonstrated with the following policy snippet:

$$\begin{aligned}
&\pi_0(Sub, Tar, Act) \Leftarrow \mathsf{t}[\dots]\\
&\pi_1(Sub, b, read) \Leftarrow \mathsf{f} \textbf{ if } \pi_0(Sub, a, read)\\
&\pi_2(Sub, a, read) \Leftarrow \mathsf{f} \textbf{ if } \pi_0(Sub, b, read)\\
&\Omega(Sub, Tar, Act) \Leftarrow \pi_0(Sub, Tar, Act) \oplus \pi_1(Sub, Tar, Act) \oplus \pi_2(Sub, Tar, Act)
\end{aligned}$$

In this case, π_1 and π_2 prevent the override to be granted by Ω when the conflict appears over a or b.

6.4.1. Composite Break-glass Rules: Compositional Expressiveness. Following the discussion regarding the compositional expressiveness of evidential rules, we can state the following theorem:

THEOREM 6.15. *Let Δ_{Π}^{BG} be a set of Π's break-glass rules, and Σ_{Π}^{BG} its set of ground break-glass atoms (with n as the highest number used in the rule identifiers). Then for a given $g : \mathcal{FOUR}^n \mapsto \mathcal{FOUR}$, there is a Beagle rule $\pi_{n+1}(sub, tar, act) \Leftarrow \psi$ such that $\textbf{\textit{IFM}}_{\Pi}(\pi_{n+1}(sub, tar, act)) = \textbf{\textit{IFM}}_{\Pi}(\psi) = g(\textbf{\textit{IFM}}_{\Pi}(p_1), \dots, \textbf{\textit{IFM}}_{\Pi}(p_n))$, where $p_i \in \Sigma_{\Pi}^{BG}$.*

PROOF. The proof follows from: (1) Theorem 6.9's proof when its set of facts is replaced with Γ_{Π}^{BG}, and (2) observing that no cyclic dependencies can exist between π_{n+1} and other rules, since π_{n+1} is a newly added predicate to $\mathcal{L}_{\Pi}^{BG}$. Thus it can be safely added to Δ_{Π}^{BG} while preserving its correctness. $\square$

This is an important result, since it means that Rumpole can express any combinatorial operator that maps decisions of break-glass rules onto an another decision.

To further illustrate how composite rules can capture various combinatorial operators for break-glass rules, we define two operators, priority and majority-rule.

The *priority override* operator, $\pi_1(Sub, Tar, Act) \rhd_\bot \pi_2(Sub, Tar, Act)$ uses its right-hand side rule only if the left-hand side rule contains a gap. We can simply use this operator as a syntactic short-hand for the following body formula:

$$\pi_1 \oplus (\top[\pi_1 = \bot] \otimes \pi_2)$$

In a similar manner, we can have $\rhd_\top$ defined as $\pi_3 \Leftarrow \pi_1 \otimes (\top[\pi_1 \neq \top] \oplus \pi_2)$.

The *majority rule* operator supports the decision made by the largest subset of rules. To this end, PBel defines an interesting majority-rule operator for three rules, $G_3(\pi_1, \pi_2, \pi_3)$, whose definition is: $(\pi_1 \wedge \pi_2) \vee (\pi_1 \wedge \pi_3) \vee (\pi_2 \wedge \pi_3)$. We can expand the operator in the same fashion to define G_4 that takes four rules as its arguments:

$$\begin{aligned}&(\pi_1 \wedge \pi_2 \wedge \pi_3) \vee (\pi_1 \wedge \pi_2 \wedge \pi_4)\\ &\quad \vee (\pi_1 \wedge \pi_3 \wedge \pi_4) \vee (\pi_2 \wedge \pi_3 \wedge \pi_4)\end{aligned}$$

There are cases when there is no majority, such as two $\top$s and two $\bot$s. With the current G_4 encoding, the decision would be f; it would be the greatest lower bound in $\leq_t$ ordering of the individual decisions. This can be further changed as needed.

6.5. Grant Policy: Determining the Final Override Decision

A grant policy is a designated composite break-glass rule that a PDP uses to make the override decision (Definition 6.5), formally:

Definition 6.16. A grant policy is a set of Beagle rules of the following form:

$$\Omega(Sub, Tar, Act) \Leftarrow \psi$$

where ψ contains only predicates from $\mathcal{L}_\Pi^{BG}$.

Theorem 6.15 establishes that a grant policy can capture any combinatorial mapping between override decisions of Π's break-glass rules. However, since primitive break-glass rules are syntactically constrained, we need to establish that these syntactic constraints over the break-glass rules do not limit a policy writer in expressing how the knowledge of the context of a break-glass request can be mapped onto override decisions and their required obligations. We want to establish that any combination of evidential and obligation atoms can be mapped onto $grant$ and $deny$ decisions. We do not have an explicit mapping for the $request\_obligations$ decision since its purpose to put forth additional obligations which ultimately result in a $grant$ decision.

We therefore need to establish that Rumpole can express all desired mappings from selected evidential and obligation atoms' evaluations onto t and that no other evaluations result in t (within the expression that captures this mapping). To this end we state the following theorem:

THEOREM 6.17. *Given* Π*'s* $\mathcal{C}_\Pi$ *and an arbitrary* $f_{\langle sub,tar,act\rangle} : \mathcal{I}(\mathcal{C}_\Pi) \mapsto \mathcal{FOUR}$ *for the request* $\langle sub, tar, act\rangle$*. Then a Beagle rule,* $\Omega(sub, tar, act) \Leftarrow \psi$*, can be added to* Δ_Π^{BG} *such that:*

$$\textbf{\textit{IFM}}_\Pi(\Omega(sub, tar, act)) = \textbf{\textit{IFM}}_\Pi(\psi) = f_{\langle sub,tar,act\rangle}(\textbf{\textit{IFM}}_\Pi(a_0), \ldots, \textbf{\textit{IFM}}_\Pi(a_n))$$

where $a_i \in \mathcal{C}_\Pi$.

This theorem establishes that a policy writer can define a grant policy as an arbitrary mapping from evidential and obligation atoms onto a required truth value from $\mathcal{FOUR}$, thus implicitly establishing that any mapping from evidential and obligation atoms onto break-glass decisions can be expressed. In this theorem, we assume that two requests are identical if they share the same request constants.

PROOF. The proof consists of two main parts. The first part shows how a set of primitive break-glass rules can be used to represent each truth-value assignment that an interpretation can make. Using this representation the task of constructing $f_{\langle sub,tar,act\rangle}$ is abstracted from $I(\mathcal{C}_\Pi)$ to a set of (newly constructed) break-glass rules. The second part uses theorem 6.15 to define a grant policy, as a composite break-glass rule, whose body captures an arbitrary $f_{\langle sub,tar,act\rangle}$ of a set of break-glass rule decisions. We now turn to a detailed description of the proof:
(1) With each ground atom $a \in \mathcal{C}_\Pi$, we associate a set of tuples (a, v) where $v \in V_\mathcal{B}$. Each tuple is assigned a unique value $i \in \mathbb{N}_0$ and a corresponding break-glass rule predicate (π_i) in $\mathcal{L}^{BG}_\Pi$. Thus, we can add a Beagle rule $\pi_i(Sub, Tar, Act) \Leftarrow \mathrm{t}$ **if** $a = v$, where i is associated with (a, v), to Δ^{BG}_Π. Therefore for each possible interpretation I there is a set of break-glass rule decisions that reflect this interpretation by assigning t to $\pi_i(Sub, Tar, Act)$ iff $I(a) = v$ (i is assigned to (a, v)). Hence the interpretation is *represented* through a set of primitive break-glass rules, i.e. each atom has a set of π_i's and only one of them has value t at any one time. In this way, the break-glass rules are only implicitly linked to an override request if a subset of request's constants (sub, act, tar) are in (ground) a which a particular rule represents.
(2) Following theorem 6.15, we can define $\Omega(sub, tar, act)$ as any function f that maps the set of π_i's onto $\mathcal{FOUR}$. For an arbitrary f we can add: $\Omega(sub, tar, act) \Leftarrow \psi$ to Δ^{BG}_Π, where $\boldsymbol{IFM}_\Pi(\psi) = f(\boldsymbol{IFM}_\Pi(\pi_0(sub, tar, act)), \ldots, \boldsymbol{IFM}_\Pi(\pi_n(sub, tar, act)))$. Thus we have $\boldsymbol{IFM}_\Pi(\Omega(sub, tar, act)) = \boldsymbol{IFM}_\Pi(\psi)$ ($\Omega(sub, tar, act)$ is the head of only one rule in Δ^{BG}_Π). Furthermore Ω can be constructed for any request since the primitive C_Π-*representative* break-glass rules are defined over all requests. □

Note that a grant policy specification is stratified since a composite policy is acyclic and evidential rules cannot use break-glass rule atoms.

6.6. Complexity of Policy Evaluation

The time complexity of making an override decision for a given set of accepted obligations is in PTIME. This follows from the complexity of Beagle's iterated fixpoint construction. The complexity of $request\_obls$ decision is in NP. This is because a grant query has a finite number of grounded obligations that can make the query *true*. Although there are not scalable decision procedures for determining required obligations, in practice if the number of obligations is low then the brute-force searches can be used.

The decision problem of *obligation accountability* is closely related to the problem of evaluating $request\_obls$ decisions. The former decides whether there exists a set of obligations that a subject can fulfill in the future, but for which he may not currently hold all the necessary permissions [Irwin et al. 2006]. This problem is in general intractable, but the authors show that with certain restrictions the problem is in PTIME. For our future work, we intend to study whether additional restrictions on Rumpole policies can similarly put the problem of evaluating $request\_obls$ in PTIME.

7. CASE STUDY: REGULATING OVERRIDES IN THE HIPAA PRIVACY RULE

In this section, we first study and highlight override provisions within the US Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule, henceforth the "HIPAA Privacy Rule". Second, we use Rumpole to construct a HIPAA-compliant break-glass policy.

Exisiting formalizations of the HIPPA Privacy Rule in [Barth et al. 2006] and [Chowdhury et al. 2013] focus on encoding HIPAA's access control provisions and not on the override provisions. Our work is complementary to these efforts since the access control provisions are enforced through AC-PEPs, and override provisions through BPEPs (see Figure 3).

7.1. Overview of the HIPAA Privacy Rule

The U.S. Department of Health and Human Services (US-HHS) issued the HIPAA Privacy Rule to implement the requirements posed by the Health Insurance Portability and Accountability Act (HIPAA) of 1996. Our overview is based on: (i) the HIPAA Rule's motivations and principles highlighted in "Summary Of the HIPAA Privacy Rule" by US-HHS [for Civil Rights 2003]; (ii) the HIPAA Rule's specification, *45 Code of Federal Regulations (C.F.R.) Part 164*, published by the U.S. Federal Register[1].

7.1.1. The HIPAA Privacy Rule's main permissions. The HIPAA Privacy Rule's main goal is: "to assure that individuals health information is properly protected while allowing the flow of health information needed to provide and promote high quality health care and to protect the public's health and well being". The HIPAA Privacy Rule regulates the use and disclosure of individuals' health information, referred to as "protected health information" (PHI) by organisations and individuals (termed "covered entities") subject to the HIPAA Privacy Rule. We focus on hospitals (as covered entities) and their rights and duties with respect to usage and disclosure of PHI.

The HIPAA Privacy Rule's key permission rule is: a covered entity may use and disclose PHI, without consulting the individual, for the following main purposes. (1) Disclosure to the individual. (2) Treatment, payment, and health-care operations (*45 C.F.R. §164.502(a)*), where treatment is defined as a provision, coordination, or management of health-care services for an individual by one or more health-care providers (*45 C.F.R. §164.506(c)*). (3) Public Interest and Benefit Activities (*45 C.F.R. §164.512*), such as: public health, research, disclosures regarding domestic violence, law enforcement, disclosures to avert a serious threat to health or safety. Furthermore, a covered entity may not use or disclose PHI if the HIPAA Privacy Rule does not permit it (*45 C.F.R. §164.502(a)*), or unless it obtains the individual's (written) authorisation (*45 C.F.R. §164.508*).

The outlined core permission is overruled in case PHI contains psychotherapy notes (*45 C.F.R §164.508*). In detail, a covered entity must obtain an individual's authorisation to use or disclose psychotherapy notes with the following main exception: they can be disclosed to avert serious threat or immediate danger to the individual.

Individuals can also request that a covered entity further restricts the use and disclosure of its PHI (*45 C.F.R. §164.522(a)*). A covered entity is not obliged to accept such a request, but if it accepts the request then it must comply with the agreed restrictions, except in the case of a medical emergency.

7.1.2. Enforcement obligations. In addition to regulating the use and disclosure of PHI, the HIPAA Privacy Rule also sets obligatory enforcements that a covered entity must put in place when guarding the access to PHI. Any HIPAA-compliant access control policy must respect the principle of "minimal necessary" use and disclosure (*45 C.F.R. §160.502(b)*). That is, when using or disclosing PHI, a covered entity must make reasonable efforts to limit PHI to the minimum necessary to fulfil its tasks. The HIPAA Privacy Rule mandates that a covered entity identifies those people in its workforce who need to use or disclose PHI during the course of their work tasks, and then it needs to identify which subset of the PHI is required for each member of its workforce (*45 C.F.R. §164.514(d)*). However, treatments and law enforcement purposes override the minimality principle. For example, a ward-nurse can override the minimality principle and access the PHI of a patient who is not under her care. But, had this nurse wanted the PHI for the purpose of research, she would have needed to obtain an explicit authorisation to do so.

[1]http://www.gpo.gov/fdsys/pkg/FR-2011-05-31/pdf/2011-13297.pdf

7.2. A Break-glass Policy for HIPAA-compliant Overrides

Even though the HIPAA Privacy Rule does not employ explicit "break-glass" and "override" terms and clauses, the given overview identifies two main situations where emergencies and threats to safety take higher priority than the regular permissions:
(1) Averting serious threat and injury – If an individual is under threat, the section §164.512(j) permits the disclosure of PHI (regardless of any restrictions), including psychotherapy notes, to a care provider. Such an override should be reviewed by an appropriate person or persons.
(2) Overriding a hospital's established minimality principle – A hospital must enforce the minimality principle by restricting access to individuals' PHI based on perceived and assumed work requirements. The details regarding enforcement of the minimality principle, such as which staff roles have which permissions, are managed internally by the hospital itself. As argued, it is impossible to predict and foresee all possible data requirements. The HIPAA Privacy Rule explicitly acknowledges this notion in §160.514(d), stating that all irregular PHI uses are reported and subsequently reviewed. Note that the overrides of the minimality principle assume that a patient is not under immediate threat. This means that the psychotherapy notes and explicitly censored information by a patient should never be released through these overrides. Furthermore, they should be restricted to treatment and law enforcement purposes which have a high priority.

Although these are not the only override situations or obligations that must be enforced within the HIPAA Privacy Rule, they are sufficiently illustrative for the purpose of our case study.

Given the identified two break-glass provisions, we propose the following requirements that a HIPAA-compliant hospital may choose to enforce:

R1 – Each override must be accompanied with a reason, and reviewed.
R2 – Patient's explicit prohibitions can be overridden only if there is an emergency. An emergency is established either by a *designated* staff member or by the patient's monitoring sensors. If it is impossible to establish an emergency, a subject can be permitted an override if additional alarms are raised and he is not explicitly prohibited.
R3 – All *minimality principle* overrides must be granted by the Head of Medicine.
R4 – All *minimality principle* overrides are granted only to senior staff members and exclude psychotherapy notes.

R1 and R2 are mandated by the HIPAA Privacy Rule, while R3 and R4 stem from our intuition for general and practical *fine-grained* override provisions that are in accordance with the HIPAA override provisions.

7.3. Specifying HIPAA Break-glass Policy in Rumpole

The approach taken for encoding a break-glass policy that enforces the given requirements consists of three steps. (1) For each requirement we define a set of break-glass rules that capture the conditions needed to enforce it. (2) These rules are combined within the grant policy to enforce the requirements. Break-glass rules provide the necessary requirements' building blocks, and the actual enforcement depends on their placement within the grant policy. (3) We define reactive obligations to use in the break-glass rules.

For *Requirement 1*, we need π_{11} and π_{12}:

$$\pi_{11}(Sub, Tar, Act) \Leftarrow \mathsf{t}\ \textbf{if}\ \mathit{acceptedObl}(sys, \textbf{\textit{Sub:Tar:Act}}, review, 36)$$
$$\pi_{12}(Sub, Tar, Act) \Leftarrow \mathsf{t}\ \textbf{if}\ \mathit{acceptedObl}(Sub, reason, submit, 24)$$

The expression *Sub:Tar:Act* is a constant, which carries all the elements of the override request. For example, a request $(alice, bob_{PHI}, read)$ corresponds to *alice:bob*$_{PHI}$*:read* constant.

For *Requirement 2*, we need π_{21}, $\pi 22$, and π_{23}:

$$\begin{aligned}\pi_{21}(Sub, Tar, Act) &\Leftarrow \mathsf{t}[emergency(Tar) \geq_k \mathsf{t}]\\ \pi_{22}(Sub, Tar, Act) &\Leftarrow \mathsf{f}[prohibited(Sub, Tar, Act) \geq_t \mathsf{t}]\\ \pi_{23}(Sub, Tar, Act) &\Leftarrow (\mathsf{t}[emergency(Tar) \leq_t \bot])\\ &\qquad\qquad \textbf{if } acceptedObl(sys, \textbf{\textit{Sub:Tar:Act}}, alert, 0)\end{aligned}$$

For *Requirement 3*, we need π_3:

$$\pi_3(Sub, Tar, Act) \Leftarrow \mathsf{t}[granted(\textbf{\textit{head\_of\_medicine}}, Sub, Tar) = \mathsf{t}]$$

For *Requirement 4*, we need π_4, π_5:

$$\begin{aligned}\pi_4(Sub, Tar, Act) &\Leftarrow \mathsf{t}[senior(Sub) = \mathsf{t})]\\ \pi_5(Sub, Tar, Act) &\Leftarrow \mathsf{t}[p\_notes(Tar) = \mathsf{t})]\end{aligned}$$

Grant policy. We now define the grant policy which ties the given break-glass rules together. We start with the enforcement of *Requirement 1*:

$$\Omega \Leftarrow (\pi_{11} \wedge \pi_{12}) \wedge \pi_{100}$$

The given rule enforces *Requirement 1* by not allowing an override grant unless the first two rules are satisfied, i.e. their obligations accepted. *Requirement 2* is enforcement with π_{100}.

$$\pi_{100} \Leftarrow (\pi_{21} \vee \pi_{23}) \triangleright_{\bot} (\pi_{22} \triangleright_{\bot} \pi_{101})$$

If either π_{21} or π_{23} are t then the override is granted. If they are not then the override is not treated as an emergency override, and π_{22} will block any prohibited requests, since it returns f, which results in $deny$. If the request is not prohibited its evaluations is delegated to π_{101} (enforcing Requirements 3 and 4):

$$\pi_{101} \Leftarrow \pi_3 \wedge \pi_4 \wedge \neg\pi_5$$

7.4. Evaluation Example

To illustrate how the specified policy is evaluated, when deployed within a system, consider the following scenario: *Patient* Bob *is not in an emergency and nurse* $Alice$ *is requesting read access to Bob's psychotherapy notes*.

Since the first two obligations are mandatory, we assume that all subjects have accepted them in advance due to their familiarity with the system. The request is represented as: $\langle alice, \textbf{\textit{bob:p\_notes}}, read\rangle$ together with the accepted obligations:

$$\begin{aligned}\Delta_{acceptedObl} = \{&acceptedObl(alice, reason, submit, 24),\\ &acceptedObl(sys, \textbf{\textit{alice:bob:p\_notes:read}}, review, 36)\}\end{aligned}$$

When $emergency(bob) = f$, the immediate decision for $\Omega(alice, \textbf{\textit{bob:p\_notes}}, read)$ is f. A Policy Decision Point (PDP) attempts to find any further obligation. Since the reading p_notes is prohibited, the PDP reports the *deny* decision. Since $Alice$ is convinced that the emergency is real, she raises the emergency herself and gets the override.

8. SUMMARY

The last decade has seen the emergence of security models, such as trust management and auditing models, that supplement traditional access control models to cope with the problem of having to predict and anticipate all permissible requests. Break-glass models are a less well known approach to handle this problem. The central idea is to permit a subject to override a denial if the subject accepts particular obligations. Existing break-glass models, however, either grant overrides for a pre-defined set of exceptional situations, or they grant unlimited overrides to selected subjects.

Rumpole is a novel break-glass language that explicitly represents and infers knowledge gaps and knowledge conflicts about a subjects attributes as well as contextual situations such as emergencies and can vary the severity of obligations based on the quality of evidence available. The formal semantics of Rumpole is based on Beagle, a novel many-valued logic programming language that extends Fittings bilattice-based semantics for logic programs.

Rumpole's override decision procedure is in PTIME for a given set of accepted obligations and contextual facts. However, the procedure to find a set obligations, which can potentially lead to a permitted override, is in NP. As our future work, we plan to explore different syntactic restrictions on obligations to result in tractable solutions for the problem of finding obligations. Another aspect of our future work is an analysis procedure for satisfiability and containment of Rumpole policies.

REFERENCES

2004. Break-glass: An approach to granting emergency access to healthcare systems. *White Paper, Joint NEMA/COCIR/JIRA Security and Privacy Committee (SPC)* (2004).

R.J. Anderson. 1996. A security policy model for clinical information systems. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. 30–43.

Claudio A. Ardagna, Sabrina De Capitani di Vimercati, Sara Foresti, Tyrone W. Grandison, Sushil Jajodia, and Pierangela Samarati. 2010. Access control for smarter healthcare using policy spaces. *Computers & Security* 29 (2010), 848 – 858.

Claudio Agostino Ardagna, Sabrina De Capitani di Vimercati, Tyrone Grandison, Sushil Jajodia, and Pierangela Samarati. 2008. Regulating Exceptions in Healthcare Using Policy Spaces. In *DBSec*. 254–267.

Ofer Arieli and Arnon Avron. 1998. The value of the four values. *Artif. Intell.* 102 (June 1998), 97–141.

A. Barth, A. Datta, J.C. Mitchell, and H. Nissenbaum. 2006. Privacy and contextual integrity: framework and applications. In *IEEE Symposium on Security and Privacy*.

N. D. Belnap. 1977. A useful four-valued logic. *Modern Uses of Multiple-Valued Logics* (1977), 8–37.

Claudio Bettini, Sushil Jajodia, X. Sean Wang, and Duminda Wijesekera. 2002. Provisions and obligations in policy management and security applications. In *Proceedings of the 28th International Conference on Very Large Data Bases*. 502–513.

Achim D. Brucker and Helmut Petritsch. 2009. Extending access control models with break-glass. In *Proceedings of the 14th ACM symposium on Access control models and technologies*. 197–206.

Glenn Bruns and Michael Huth. 2008. Access-Control Policies via Belnap Logic: Effective and Efficient Composition and Analysis. *IEEE Computer Security Foundations Symposium* (2008), 163–176.

J. G. Cederquist, Ricardo Corin, M. A. C. Dekker, Sandro Etalle, J. I. den Hartog, and Gabriele Lenzini. 2007. Audit-based compliance control. *Int. J. Inf. Sec.* 6, 2-3 (2007), 133–151.

S. Ceri, G. Gottlob, and L. Tanca. 1989. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering* 1 (1989), 146–166.

Omar Chowdhury, Andreas Gampe, Jianwei Niu, Jeffery von Ronne, Jared Bennatt, Anupam Datta, Limin Jia, and William H. Winsborough. 2013. Privacy Promises That Can Be Kept: A Policy Analysis Method with Application to the HIPAA Privacy Rule. In *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*. 3–14.

Jason Crampton and Charles Morisset. 2012. PTaCL: A Language for Attribute-Based Access Control in Open Systems. In *Conference on Principles of Security and Trust (POST)*. 390–409.

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33 (September 2001), 374–425.

Sandro Etalle and William H. Winsborough. 2007. A posteriori compliance control. In *SACMAT*. 11–20.

A. Ferreira, D. Chadwick, P. Farinha, R. Correia, Gansen Zao, R. Chilro, and L. Antunes. 2009. How to Securely Break into RBAC: The BTG-RBAC Model. In *Computer Security Applications Conference*.

Melvin Fitting. 1990. Bilattices in logic programming. In *Multiple-Valued Logic*. 238–246.

Melvin Fitting. 1991. Bilattices and the semantics of logic programming. *J. Log. Program.* 11 (1991), 91–116.

Office for Civil Rights. 2003. Summary of the HIPAA privacy rule. *United States Department of Health & Human Services* (2003).

Yann Le Gall, Adam J. Lee, and Apu Kapadia. 2012. PlexC: a policy language for exposure control. In *SACMAT '12*. 219–228.

Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. 1991. The Well-Founded Semantics for General Logic Programs. *J. ACM* 38, 3 (1991), 620–650.

Matthew Ginsberg. 1988. Multivalued Logics: A Uniform Approach to Inference in Artificial Intelligence. *Computational Intelligence* 4 (1988), 265–316.

S. K. S. Gupta, T. Mukherjee, and K. Venkatasubramanian. 2006. Criticality Aware Access Control Model for Pervasive Applications. In *PERCOM '06*. IEEE Computer Society, 251–257.

Ragib Hasan and Marianne Winslett. 2011. Efficient audit-based compliance for relational data retention. In *ASIACCS*. 238–248.

Keith Irwin, Ting Yu, and William H. Winsborough. 2006. On the Modeling and Analysis of Obligations. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*. 134–143.

Adam J. Lee, Jodie P. Boyer, Lars E. Olson, and Carl A. Gunter. 2006. Defeasible Security Policy Composition for Web Services. In *Proceedings of the Fourth ACM Workshop on Formal Methods in Security*. 45–54.

Ninghui Li, Qihua Wang, Wahbeh Qardaji, Elisa Bertino, Prathima Rao, Jorge Lobo, and Dan Lin. 2009. Access control policy combining: theory meets practice. In *SACMAT '09*. ACM, 135–144.

Jim J. Longstaff, Mike A. Lockyer, and M. G. Thick. 2000. A model of accountability, confidentiality and override for healthcare and other applications. In *ACM Workshop on Role-Based Access Control*. 71–76.

Srdjan Marinovic, Robert Craven, Jiefei Ma, and Naranker Dulay. 2011. Rumpole: a flexible break-glass access control model. In *SACMAT*. 73–82.

Qun Ni, Elisa Bertino, and Jorge Lobo. 2008. An obligation model bridging access control policies and privacy policies. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM, New York, NY, USA, 133–142.

Jaehong Park and Ravi Sandhu. 2004. The UCONABC usage control model. *ACM Trans. Inf. Syst. Secur.* 7, 1 (2004), 128–174.

Dean Povey. 2000. Optimistic security: a new access control paradigm. In *NSPW '99: Proceedings of the 1999 workshop on New security paradigms*. ACM, New York, NY, USA, 40–45.

Teodor C. Przymusinski. 1988a. On the Relationship Between Logic Programming and Nonmonotonic Reasoning. In *AAAI*. 444–448.

Teodor C. Przymusinski. 1988b. Perfect Model Semantics. In *ICLP/SLP*. 1081–1096.

R. Reiter. 1977. *On Closed World Data Bases*. Technical Report. Vancouver, BC, Canada.

Erik Rissanen, Babak Sadighi Firozabadi, and Marek J. Sergot. 2004. Discretionary Overriding of Access Control in the Privilege Calculus. In *Formal Aspects in Security and Trust*. 219–232.

Paul Ruet and François Fages. 1997. Combining explicit negation and negation by failure via Belnap's logic. *Theor. Comput. Sci.* 171 (January 1997), 61–75.

John S. Schlipf. 1995. Complexity and undecidability results for logic programming. *Annals of Mathematics and Artificial Intelligence* 15 (1995), 257–288.

V. s. Subrahmanian. 1999. Nonmonotonic Logic Programming. *IEEE Trans. on Knowl. and Data Eng.* 11, 1 (1999), 143–152.

Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.* 5, 2 (1955).

K. Twidle, E. Lupu, N. Dulay, and M. Sloman. 2008. Ponder2 - A Policy Environment for Autonomous Pervasive Systems. In *POLICY 2008*. 245–246.