ABSTRACT

DABRAL, CHINMAYA. Explaining Privacy Policies through Narrative Generation. (Under the direction of Dr. Arnav Jhala).

Privacy policies and laws encode precise, logic-like requirements, yet they are communicated through lengthy natural-language documents. This mismatch makes it difficult for companies, regulators, and end-users to interpret and apply the underlying rules after extracting the logical structure from dense legal prose.

Policy formalization systems address part of this challenge by encoding policies as logical rules that can be executed for tasks such as compliance checking. However, these formal models remain inaccessible to many privacy practitioners and do not naturally capture the concrete, scenario-based situations that policies are intended to regulate. Our goal is to explore a bridge between formal policy modeling and scenario-based representations by encoding policy rulesets in the context of a narrative planner.

This thesis presents a novel approach combining formal policy modeling and narrative generation within a logical programming framework, yielding a flexible system that allows many modes of interaction with policy rulesets: generating scenario-based "what-if" narratives, tracing compliance-relevant event sequences, and understanding policy intent by examining the space of permitted and prohibited narrative outcomes. In addition, we introduce a semi-automated method for constructing planning domains from natural-language policy text.

Explaining Privacy Policies through Narrative Generation

by
Chinmaya Dabral

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2025

APPROVED BY:

Dr. Anupam Das	Dr. Collin Lynch
Dr. Ruozhou Yu	Dr. Arnav Jhala Chair of Advisory Committee

BIOGRAPHY

Chinmaya Dabral earned his Bachelor of Engineering with Honors from Birla Institute of Technology and Science, Pilani and pursued his Ph.D. in Computer Science at North Carolina State University under the advisement of Dr. Chris Martens and later Dr. Arnav Jhala. His doctoral research primarily focused on narrative generation, logic programming, and policy formalization. His other interests include programming languages and slicing algorithms for 3D printing.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor Dr. Arnav Jhala and my former advisor Dr. Chris Martens, whose guidance, patience, and encouragement were invaluable throughout the course of my research. Iam grateful to my committee members, Dr. Anupam Das, Dr. Collin Lynch and Dr. Ruozhou Yu for their thoughtful feedback on my research.

Special thanks go to my brother, Tanmaya, for always being there for everything. Same to my roommates Asmita Narode and Abhijeet Krishnan, and friend and labmate Sasha Azad and everyone else for fun gatherings and stimulating conversations. I'd also like to thank all of the above for helping me through two major surgeries and rehab, along with the skilled surgical teams at Duke Hospital who made it possible for me to continue my research.

And lastly, I'd like to thank the four stray kitties, Plum, Pineapple, Apple, Maple, and their mom Smoothie who gave birth to them in my patio. Even on the most difficult days, I could look forward to a spark of delight at the end of the temporal tunnel as they returned every night to eat, play, and sleep. I wish you knew how much I owe you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Thesis statement: *A planning-based approach serves as a viable alternative to textual question answering systems in the policy domain, while enabling richer interaction with the solution space.*

The need to understand complex rulesets is all around us, including those governing board games and sports, end-user software policies, the law defined by government, product user manuals, and scientific theories. We posit that explorability has a critical role to play in making sense of such rule systems. Ideally, rulesets could be understood not as static documents, but as changing, interactive worlds, in which questions can be answered with specific examples, hypotheses tested and refined, and the consequences of pertinent scenarios be explored.

Narrative generation plays a key role in realizing this vision. Humans have been shown time after time to make better sense of complex information through *stories*, where a progression of events over time with clear cause and effect relationships is depicted, than through unorganized collections of facts (Bruner 1991; Gerrig and Wenzel 2015). Therefore, narrative generation is a key competency that such a system must demonstrate. Existing approaches based on planning offer a promising path forward, due to their explicit world models that represent granular state change.

The overarching goal of this thesis is to realize this vision of explorable formal models, focusing on privacy policies and regulations as a timely case study.

The structure of this thesis corresponds roughly to our research questions:

RQ 1: *How can we automatically identify privacy incidents from unstructured online text sources to build a database that captures privacy violations beyond traditional security breaches?*

RQ 2: *Can we design a narrative planner that supports:*

1. *Encoding of policies as narrative constraints*
2. *Encoding of arbitrary rulesets*

RQ 3: *How do we encode the privacy-specific knowledge contained in real-world policies and regulations in the context of a narrative planner?*

RQ 4: *To what extent can we automate encoding of policy domains as narrative constraints?*

Chapter 2 identifies a gap in knowledge, with no existing databases capturing the full spectrum of privacy incidents. We address this by developing an incident classifier based on 1324 manually coded news articles.

Chapter 3 presents a narrative planner written in Answer Set Programming (ASP), capable of supporting arbitrary rulesets, including policy formalizations, as constraints on the narrative space.

Chapter 4 presents a methodology to encode policies as narrative domains, using data transmission rules of HIPAA (Health Insurance Portability and Accountability Act) as a case study.

Chapter 5 investigates usage of LLMs (Large Language Models) to assist in encoding natural language policy text as narrative domains for our planner.

CHAPTER

2

BUILDING A PRIVACY INCIDENTS DATABASE

RQ 1: *How can we automatically identify privacy incidents from unstructured online text sources to build a database that captures privacy violations beyond traditional security breaches?*

2.1 Authorship Statement

This chapter contains work undertaken in collaboration with Dr. Jessica Staddon's research group as a joint research effort, presented as Murukannaiah et al. (2017), with me as the second author. My primary contributions to the work involved collection and sanitization of the Guardian part of the dataset and manual grading of part of the overall dataset. I jointly contributed to feature engineering discussions and classifier experiments. The conceptual framework was led by Dr. Murukannaiah and Dr. Staddon.

2.2 Introduction

Many databases exist for *security* incidents. Indeed, the patterns and characteristics of security incidents, as captured by these databases, are a significant driver of security technology innovation. Patterns are detected by analyzing repositories of malware/viruses/worms (e.g. (Enigma Software Group; Microsoft; Symantec)), incidents affecting control/SCADA systems (Security Incidents Organization), general security alerts and updates (US-CERT) and data breaches (e.g., (PRC)). These databases have advanced the science of security. For example, the malware database VX Heavens (Cluley March 28, 2012) is referenced in almost 8,000 research papers according to Google Scholar, many of which tested algorithms on data supplied by VX Heavens before it was shut down in 2012.

Although there is some overlap between privacy and security incidents, most types of privacy incidents are not represented in these security incident repositories. In particular, incidents of cyber-bullying/stalking, revenge porn, social media oversharing, data reidentification and surveillance, generally do not involve a security incident and so are not included in the current repositories. Table 2.1 demonstrates the diversity of privacy incident types.

Even for areas in which security incident databases include privacy incidents (e.g. data breaches), analysis is difficult. Current databases are not synchronized, making it difficult to compare across them and calculate accurate statistics. For example, for the year 2014, the Privacy Rights Clearinghouse database (PRC) finds less than 400 data breach incidents (Huq September, 2015), Romanosky finds approximately 1200 (Romanosky January, 2016), and, using proprietary data from 70 companies and organizations, Verizon finds over 2000 breaches (Verizon 2015). The difficulty of tracking the frequency and consequences of data breaches in the absence of a comprehensive database was noted as early as 2007 in a U.S. Government accountability report (U. S. Government Accountability Office 2007).

While a complete analysis requires a comprehensive database, even an ad hoc incident analysis indicates that there are common elements of incidents that, when identified, improve system privacy. For example, a number of Internet companies have introduced new privacy policies and user consent approaches that have received negative reactions from end users or regulators. Privacy policy changes at Spotify (A. Mamiit. September 5, 2015) and Google (Article 29 Data Protection Working Party September 2014), and user consent mechanisms at Facebook (FTC), have all been criticized as having deficiencies in user comprehension, visual presentation and user-utility. Analysis of these and other consent-related privacy incidents may suggest design patterns that diminish the chances of similar incidents going forward. For example, criticisms of "dark design patterns" (H. Brignull, M. Miquel, J. Rosenberg, J. Offer.) like these has led the FTC to specify user interface requirements for disclosures, including that they be made in "print that contrasts highly with the background on which they appear" (FTC), and motivated the Article 29 Working Party to recommend that privacy policies be "immediately visible and accessible, for instance visible without scrolling and accessible via one click, from each service landing page" (Article 29 Data Protection Working Party September 2014).

Incident patterns may also suggest engineering improvements. Recent years have seen repeated incidents of accidental sharing of personally identifiable information (PII), occurring at government agencies (A. Southall. November 6, 2010), healthcare providers (D. Palmer. September 2, 2015), online retailers (England September 2, 2015) and pharmaceutical companies (FTC January 18, 2002). Analyzing such incidents may suggest better technological approaches to detecting sensitive data before they are shared.

We are building the first comprehensive database of privacy incidents. While the potential to identify the trends and patterns described above using an incident database is a huge data mining opportunity, building such a database is a substantial data mining challenge, in itself, in terms of both efficiently gathering past privacy incidents and sustaining incident coverage going forward. To address this challenge we have manually evaluated over 1300 articles to build positive and negative training sets from which to learn a privacy incidents classifier. Since our high-dimensional data is vulnerable to classifier over-fitting, we use the information-theoretic measure of mutual information to reduce the feature set. Our best, SVM-based, classifier, achieves precision and recall that are both significantly better than keyword-based classifiers that flag articles as privacy incident-related if they contain "privacy" or related keywords. This classifier thus greatly reduces the human-review time needed when news articles are used to maintain and augment a privacy incidents database over time.

Related Work

As mentioned earlier, privacy incidents involving the disclosure of sensitive personal, financial and health information (e.g. social security numbers) have been collected by several organizations including the Privacy Rights Clearinghouse (PRC), Verizon (Verizon 2015), the Identity Theft Resource Center (ITRC), and Advisen (e.g., (Advisen September, 2015)). Existing analyses of data breaches include the consideration of organization response to breaches and the cost of such breaches (e.g., (Acquisti et al. 2006; Romanosky January, 2016; Shey, Heidi.)), visualizations of breach data (inf), and, more recently, the consumer perspective on breach notifications (Ablon et al. 2016).

Data breaches as collected and analyzed in these works, involve the breach of a company database storing such data for multiple users, often due to a security attack. However, breaches that do not involve a company or organization database (e.g. wifi payload collection (E. Bailey Jr. June 8, 2010)) and non-breach privacy incidents are not documented by such repositories.

The privacy research community is increasingly focused on gathering and analyzing privacy incidents (e.g., (Garfinkel and Theofanos February, 2016.; Romanosky January, 2016)). Our work supports those efforts by providing an efficient means for gathering a repository of privacy incidents.

Finally, we note that this paper overlaps with and extends an unpublished working paper (P. K. Murukannaiah, J. Staddon, H. Lipford and B. Knijnenburg.).

2.3 Vision and Prototype

We adopt a deliberately broad definition of "privacy incident", in particular, as *an event involving accidental or unauthorized collection, use or exposure of sensitive information about an individual,* ***or*** *an event that creates the perception that unauthorized collection, use or exposure of sensitive information about an individual may happen or is happening,* ***and*** *the event involves data in digital form.*

With this definition we choose to include both realized privacy "harms" (e.g., as discussed in (Calo 2011)) and perceived or expected harms. The latter are important to include as they indicate policies, product features or practices that are disliked or misunderstood by users.

As an example of the importance of including perceived privacy risk, consider the revamped privacy policy introduced by the streaming music service Spotify in August 2015. The new policy included vague language about data usage and sparked a lot of criticism, leading to a reversal in September of 2015 (A. Mamiit. September 5, 2015). There is no evidence any concrete privacy harm happened while this transitory privacy policy was in place, but under our definition it is a privacy incident because it was perceived as harmful and is therefore a useful data point toward understanding users' privacy preferences and perceptions, improving policies and based on that understanding.

At a high level, a privacy incident involves end user(s), one or more software systems and, potentially, adversarial user(s). These entities may cause an incident in several ways including: a system error (e.g. a bug or misconfiguration), a user error (e.g. accidental disclosure), system design (e.g. poorly chosen default settings), a system misuse or unintended use (e.g. unwanted personalization features), and a system abuse or attack (e.g. a hacking incident or a man-in-the-middle attack). These causes may overlap. For example, a design flaw may lead to user errors, and an attacker may exploit system misconfiguration. One goal of the incident database is to better understand the frequencies of and correlations between these causes. Table 2.1 demonstrates the diversity of privacy incidents.

One goal of the database is to support trend identification, in particular, correlation of privacy incidents with external antecedents and consequences. For example, (Acquisti et al. 2006) show that data breaches have a negative impact on a company's market value on the announcement day of the breach. (Romanosky et al. 2011) show that the adoption of data breach disclosure laws have significantly reduced identity theft caused by data breaches. The privacy incidents database will markedly reduce the data collection effort required to conduct such investigations.

Because many of these incidents details only emerge over time, and may be spread out over many articles, we seek to automate the identification of articles about any privacy incident, rather than just incidents that are new to the database.

The complexity of incident root causes, as well as variation in the domains of incidents, the entities involved, and the populations impacted, argues for a collaboratively maintained database. We envision a Wikipedia-style space in which contributors can suggest incidents, gather incident evidence and reach consensus on incident attributes. The geographical range of privacy incidents suggests that the database will benefit from diverse contributors and the interdisciplinary aspect of privacy means a wide variety of communities must be engaged in maintaining the database.

2.4 Semi-Automated Incident Identification From News

Information about privacy incidents is available from a variety of sources such as news articles, blogs, and social media. However, finding privacy incidents manually, e.g., via ad hoc keyword search, is nontrivial due to the vast amount of information available on the Web and high potential for false positives when relying on keyword search to identify incidents. For example, a search for the keyword "privacy" would flag articles such as, "Johnny Manziel asks for privacy as he enters rehab" (Lutz February 2, 2015), and "Park visitors disturb privacy

Table 2.1: Examples of privacy incidents demonstrating the variety of root causes and other characteristics, bolded for readability. Incidents are grouped by predominant cause.

Root Cause	**Entities**	**Date First Known**	**Description**
System Error	Facebook LiveJournal, Xanga, Digg MySpace, Hi5	5/2010	Code used by the services allows **referrer** links containing **user IDs** to be sent to **advertisers** when users click on ads, thus identifying them. (E. Steel and J. Vascellaro.)
User Error	Kazaa	6/2002	Many Kazaa users found to be **sharing personal files** on the Kazaa network. (Good and Krekelberg 2003)
System Design	Fitbit	7/2011	**Default privacy setting** allows FitBit profiles to be surfaced by search engines, **revealing sexual activities** of FitBit users. (L. Rao. July 3, 2011)
Unintended Use/ System Misuse	Netflix	12/2009	Researchers **de-anonymized** Netflix prize dataset, causing a **closeted gay mom** to be outed based on her Netflix viewing pattern. (R. Singel.)
System Misuse/ Attack	FBI	1/2010	FBI found to have improperly gained access to **calling records of citizens**. In some cases, reporters were targeted as part of leak investigations. (Savage January 20, 2010)

of animals during mating season" (Warwick November 10, 2015), neither of which meet our definition of privacy incident, as the former is primarily concerned with news about a celebrity entering a hospital and the latter does not concern the privacy of humans.

We posit that information about privacy incidents has distinguishing features such as keywords, entities, and sentiment. Our objective is to develop automated techniques that exploit such features to enable efficient and large-scale identification of articles related to privacy incidents. Figure 2.1 provides an overview of our vision of a semi-automated framework for identifying privacy incidents. The crux of the framework is the *incident classifier*, a machine-learned model, that automatically classifies a given piece of information as related to a privacy incident or not. Further, the framework is semi-automated in the sense that an expert (human user) reviews the incidents identified by the classifier before they are added to our database.

We identify the key challenges in realizing an incident classifier as:

(1) curating a *training set*, consisting of both articles that are and are not related to privacy incidents;

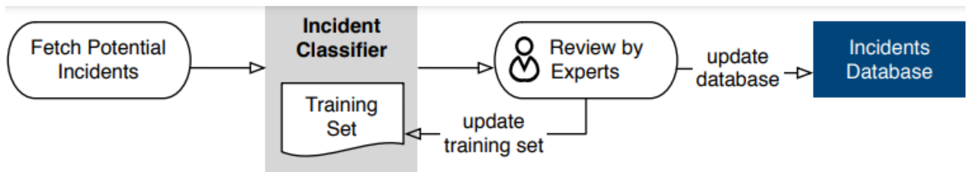

Figure 2.1: Our vision of a semi-automated technique for populating the privacy incidents database.

(2) engineering *features* to represent training instances in a vector space; and

(3) training a *classifier* to distinguish privacy incidents from nonincidents.

2.4.1 Training Set

As mentioned earlier, we focus on news articles as the source of information for identifying information related to privacy incidents. Many leading news agencies have public APIs (application programming interface), e.g., (Guardian; NYTimes), providing programmatic search and access to the news they publish, both current and historical.

Given a news article, our objective is to classify the article as either primarily concerned with one or more privacy incidents or not concerned with any privacy incidents. To do so, we develop an incident classifier via supervised machine learning . A supervised model requires a curated training set: a set of news articles in which each article is labeled as either primarily concerned with a privacy incident(s) (positive example, included in the set, P) or not (negative example, included in the set N). To make this determination, we rely on the definition of a privacy incident from Section 2.3.

To form the initial pool of articles from which P and N were extracted, we employed the following three data gathering methods. First. we randomly selected articles using the New York Times (NYTimes) and Guardian (Guardian) APIs, expecting these articles, largely, to be part of the set N (since most news is unrelated to privacy). Second, we used the keyword "privacy" and keywords found to be closely associated with privacy news (Sheshadri et al. 2016) to retrieve articles via the APIs. Third, we gathered articles that had been manual tagged as privacy articles by the New York Times. We also manually reviewed articles to identify examples that either narrowly failed to meet our definition for subtle reasons (e.g., articles regarding physical, non-digital, privacy incidents and incidents regarding security but not privacy) and articles that represented particular privacy issues falling in the Solove categories (Solove January 2006).

From the initial pool of 1104 articles gathered by these methods, we extracted 4 sets of 100 and 1 set of 198 articles. For each set, 2 coders independently reviewed the articles and coded them as either closely related to a privacy incident or unrelated to any privacy incident. Overall, the inter-coder agreement was high, with an average Cohen's Kappa (Viera and Garrett 2005) of 0.9608. Given this high agreement, we relied on a single coder for the remaining 506 articles.

For those articles reviewed by two coders, each article found as related to a privacy incident by both coders, was put in set P; similarly each article that both coders agreed as not related to privacy incidents was added to N. Any article on which there was a disagreement was not used in training set. Similarly, articles reviewed by a single coder were assigned to P or N based on the code. Finally, we also included 249 articles that had already been gathered by the Privacy Incidents Database project (NCSU et al.; P. K. Murukannaiah, J. Staddon, H. Lipford and B. Knijnenburg.), as part of P. We did not code these 249 articles since they were already reviewed by the Privacy Incidents Database project and determined to be positive examples.

As a result of this process and some de-duplication, P consists of 543 articles and N consists of 781 articles (total of 1324 out of the original pool of 1353). The most common publishers in our training data are the New York Times and the Guardian. However, the final training set ($P \cup N$) also includes a variety of other news publishers and tech blogs.

2.4.2 Feature Engineering

Our training set consists of the textual contents of a set of news articles. In order to train the incident classifier, we need to represent each news article as a set of *features*. Figure 2.2 summarizes the steps we followed to do so.

2.4.2.1 Text Preprocessing

We employ techniques from natural language processing to preprocess our dataset. First, we extract all sentences in a news article and perform *parts-of-speech* (PoS) tagging. We retain only content words (nouns, verbs, adjectives, and adverbs) and further remove stopwords (Ranks NL), based on the intuition that only the remaining words help classification. Next, we perform *lemmatization* to reduce inflectional forms of words to their base or dictionary forms (known as *lemma*), e.g., the words "collecting" and "collected" are reduced to their root form "collect." Lemmatization helps reduce the number of words in an article, and consequently, the number of features used for classification.

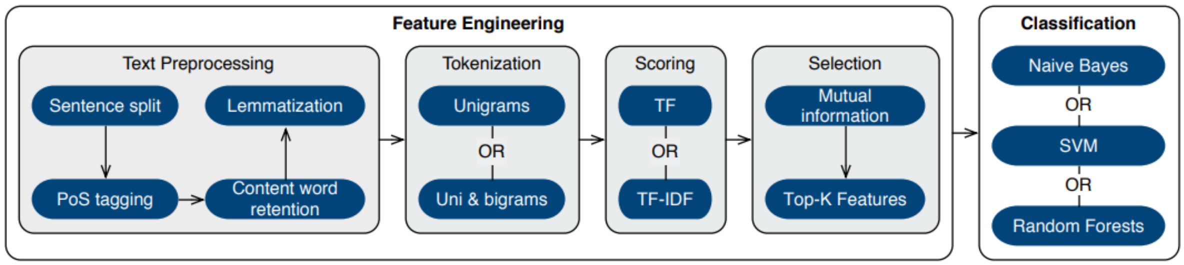

Figure 2.2: An overview of the steps we follow to engineer features from news articles for performing classification. The steps marked as OR represent some of the variations we consider.

2.4.2.2 Unigrams and Bigrams

After preprocessing, each news article consists of the lemmas of content words in that article. Next, we extract two types of tokens. A *unigram* is a unique lemma across all articles in the dataset. A *bigram* is a unique pair of consecutive lemmas across all articles in the dataset. Table 2.2 shows the number of tokens in our dataset before and after preprocessing steps.

Table 2.2: Number of tokens in our dataset.

Unique words (before preprocessing)	59,864
Unigrams (after preprocessing)	34,537
Bigrams (after preprocessing)	449,988

2.4.2.3 TF and TF-IDF Scores

Our next task is to represent each news article as a feature vector. To do so, we treat each token (unigram and/or bigram) in the entire dataset as a feature. To compute feature values, we employ the TF scores or TF-IDF scores (Manning et al. 2008, Chapter 6) described below. Note that although there are multiple variants of these scores, we employ one set of commonly used variants to demonstrate our overall approach.

The *term frequency* (TF) is defined as:

$$\mathrm{TF}(t,a) = \begin{cases} 1 + \log f_{t,a} & \text{if } f_{t,a} > 0 \\ 0 & \text{Otherwise,} \end{cases} \tag{2.1}$$

where $f_{t,a}$ is the frequency of the token t in article a. The logarithm of raw frequency is used for sublinear scaling with the intuition that a term occurring ten times in an article may not be ten times as important as a term occurring once.

Next, the *inverse document frequency* (IDF) is defined as:

$$\mathrm{IDF}(t,A) = \log \frac{N}{|a \in A : t \in a|}, \tag{2.2}$$

where t is a token in article a, A is the set of all articles in the dataset, N is the size of A, and $|a \in A : t \in a|$ is the number of articles in which t appears.

Finally, the TF-IDF score is the product of term frequency (Equation 2.1) and inverse document frequency (Equation 2.2):

$$\text{TF-IDF}(t,a,A) = \mathrm{TF}(t,a) \times \mathrm{IDF}(t,A). \tag{2.3}$$

In a nutshell, the TF-IDF score indicates the importance of a token in an article, considering all articles in the dataset. That is, for a given token, the TF-IDF score increases as the token appears more frequently in the article, but decreases as the token appears in more articles.

2.4.2.4 Feature Selection

Our dataset consists of 1324 instances (positive and negative examples combined), but a substantially larger number of features—considering only unigrams yields 34,537 features; considering bigrams, in addition, increases the number of features more than ten fold. A dataset in such a high-dimensional space poses two key challenges. First, training a classifier can be inefficient. Second, and more importantly, a classifier trained in a high-dimensional space can be prone to overfitting. That is, given a large number of features, it is likely that some of them are "rare" and exist only in the training data. Such features may yield a classifier highly accurate on the training data, but error-prone on new data.

To address these challenges, we select a subset of features before performing classification. Specifically, we employ the expected *mutual information* (MI) of a token t and a class c (Manning et al. 2008, Chapter 13) to determine whether to select a feature or not. MI is defined as:

$$\mathrm{MI}(U;C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} \mathrm{P}(e_t, e_c) \times \log_2 \frac{\mathrm{P}(e_t, e_c)}{\mathrm{P}(e_t) \times \mathrm{P}(e_c)}, \tag{2.4}$$

where U is a random variable, taking the values of $e_t = 1$ for articles containing token t and $e_t = 0$ for articles not containing t; and C is a random variable, taking values of $e_c = 1$ for articles in class c (articles related to privacy incidents) and $e_c = 0$ for articles not in c (articles not related to privacy incidents).

In a nutshell, the expected mutual information of a token t and class c measures the extent to which the presence of t contributes to classifying an article as belonging to class c. Thus, we rank all features by their mutual information values and select top K features. Table 2.3 shows top 20 features in our dataset ordered by mutual information.

Table 2.3: The top 20 features in our dataset ordered by their MI values (features with an underscore are bigrams).

Token	MI	Token	MI
privacy	0.3531	datum	0.2280
information	0.2225	company	0.1278
collect	0.0949	personal_information	0.0880
personal	0.0847	phone	0.0846
breach	0.0834	use	0.0824
access	0.0801	user	0.0753
customer	0.0736	privacy_policy	0.0712
address	0.0710	surveillance	0.0709
request	0.0621	law	0.0610
investigation	0.0575	print_april	0.0560

2.4.3 Classification

In Section 2.4.2, we transformed news articles in natural language to data instances in vector space. These data instances consist of two non-overlapping sets: news articles related to privacy incidents and those not related to privacy incidents. We refer to these two sets of articles as *privacy* class and *non-privacy* class, respectively.

2.4.3.1 Classifiers

We incorporate *classifiers* to identify whether a news article belongs to the the privacy or the non-privacy class. We learn the parameters of the classifier considering articles from the sets P and N in Section 2.4.1 as privacy and non-privacy classes, respectively, using the feature engineering of Section 2.4.2. We exploit one of the following well-known classifiers.

Naive Bayes (NB) (Pang-Ning et al. 2006, Chapter 5) is a probabilistic classifier that exploits Bayes' theorem to compute the probability of an instance belonging to a class. To estimate class-conditional probabilities, NB makes a strong (naive) assumption that features are conditionally independent of each other given the class.

Support Vector Machines (SVM) (Pang-Ning et al. 2006, Chapter 5) constructs a hyperplane to separate positive and negative data instances. Training an SVM model involves learning the parameters of a hyperplane that maximizes the *margin* between classes (informally, the margin is the extent of separation between the classes given the hyperplane). We employ SVM with a linear kernel.

Random Forests (RF) (Breiman 2001) operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes of the individual trees. Each tree is trained in a greedy fashion on a randomly chosen subset of the training data, selecting the TF-IDF feature that maximises information utility. We employ a standard RF algorithm trained on 400 trees each of depth 20 (these parameters were refined experimentally).

2.4.3.2 Baselines

We develop keyword-based classifiers to serve as baselines for evaluating the other classifiers above.

A keyword-based classifier identifies a news article as privacy or non-privacy depending on the presence or absence of certain keywords, respectively. Identify a set of keywords is the crux of this approach. Then, we predict a news article as belonging to the privacy class if it contains one of the keywords identified, and as non-privacy class, otherwise. To identify the set of keywords we employ one of the following techniques.

Privacy keyword technique employs the term "privacy" and its inflectional forms as the set of keywords.

Privacy and Solove keywords technique employs the term "privacy" and in addition, terms based on the names of Solove categories and subcategories (Solove January 2006) as the keyword set. Table 2.4 shows the Solove keywords we add and their counts in the privacy class (we also include the inflectional forms of these words, but do not show them in the table, for brevity).

Top-K keywords technique employs K most frequent tokens in the privacy class as the set of keywords. We perform the text preprocessing steps described in Section 2.4.2.1 on the articles in the privacy class before identifying the keywords. Here, we only consider nouns based on our observation that other frequent content words may not to be useful for classification (e.g., the verb "say" was the most frequent content word in the privacy class, but will likely not influence classification). Table 2.5 shows Top 20 keywords in the privacy class of our dataset.

Table 2.4: Counts of keywords representing Solove's categories and subcategories for articles in the privacy class.

Token	Count	Token	Count
		Info. dissemination	24
Info. collection	935	Breach confidentiality	694
Surveillance	545	Disclosure	289
Interrogation	3	Exposure	162
Info. processing	161	Increased accessibility	38
Aggregation	34	Blackmail	21
Identification	354	Appropriation	72
Insecurity	15	Distortion	3
Secondary use	2	Invasion	66
Exclusion	21	Intrusion	118
		Decisional interference	27

2.5 Evaluation

As Section 2.4.2 describes, our feature engineering pipeline consists of several variation points. First, we perform a comprehensive evaluation to identify an optimal set of steps.

Table 2.5: The 20 most frequent tokens (nouns only) in our dataset, considering articles in the privacy class.

Token	Count	Token	Count
datum	2349	information	2023
privacy	1897	company	1838
security	1440	user	1330
government	1261	people	1089
google	1077	facebook	1005
apple	992	law	876
app	858	court	806
case	805	service	778
year	772	phone	735
site	689	time	667

Then, we compare the performance of our incident classifier with those of the baselines. We evaluate the performance of a classifier via precision, recall, and F_1 scores defined below.

$$\begin{gathered} \text{precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}, \quad \text{recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}, \\ F_1\text{-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}, \end{gathered} \tag{2.5}$$

where TP, TN, FP, and FN refer to true and false positives and negatives.

Our dataset consists slightly more negative instances (781) than positive instances (543). We randomly excluded 238 ($= 781 - 543$) negative instances to have a balanced training set of 543 positive and negative instances, each. All results we report in this section are based on ten-fold cross-validation on the balanced training set.

2.5.1 Feature Engineering

After text preprocessing, we have options of

1. unigram vs. unigrams and bigrams;
2. TF vs. TF-IDF scores; and
3. different K values for selecting top-K features.

Figure 2.3 and Table 2.6 compare the performances of classifiers trained with features engineered via different combinations of these three options. For the values K, considering the large number of possibilities, we experiment in powers of two (1, 2, 4, 8, ... up to maximum number of features). We make three major observations from these results.

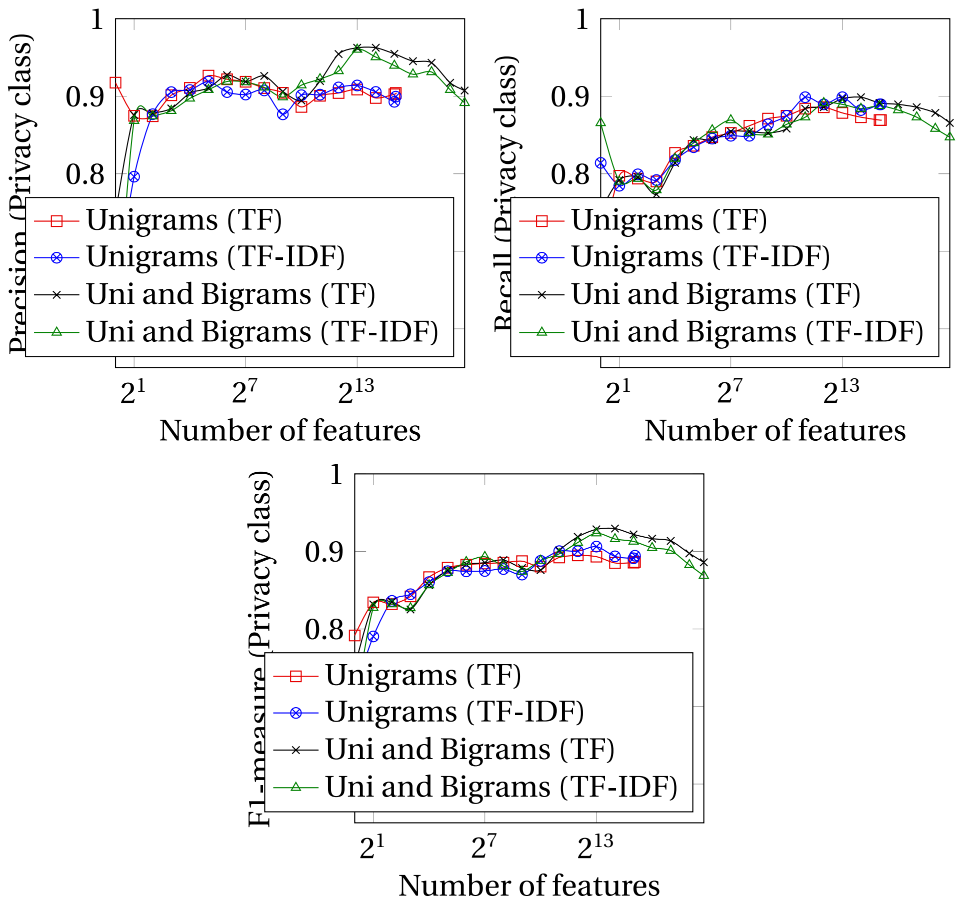

Figure 2.3: Comparing SVM classifiers trained on different top-K features selected based on information gain.

First, the choice of TF-IDF scores against TF scores makes little difference to performance. Indeed, using TF scores yields slightly better results than using TF-IDF scores.

Second, employing both unigrams and bigrams yields better results than employing only unigrams. In particular, the difference in precision is quite noticeable. This suggests that using bigrams yields certain features whose values are unique to a class, whereas the values of their constituent unigrams are not unique to a class. For example, the bigram

Table 2.6: Performance of an SVM-based classifier, considering different feature engineering techniques.

Feature		Privacy Class			Non-privacy Class			Mean		
Type	Count	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1
Unigrams (TF)	34536 (all)	0.904	0.869	0.886	0.874	0.908	0.891	0.889	0.889	0.889
Unigrams (TF)	4096 (optimal)	0.904	0.886	0.895	0.888	0.906	0.897	0.896	0.896	0.896
Unigrams (TF-IDF)	34536 (all)	0.899	0.890	0.895	0.891	0.901	0.896	0.895	0.895	0.895
Unigrams (TF-IDF)	8192 (optimal)	0.914	**0.899**	0.906	0.900	0.915	0.908	0.907	0.907	0.907
Uni & Bigrams (TF)	449987 (all)	0.907	0.866	0.886	0.871	0.912	0.891	0.889	0.889	0.889
Uni & Bigrams (TF)	8192 (optimal)	**0.962**	0.897	**0.929**	**0.903**	**0.965**	**0.933**	**0.933**	**0.931**	**0.931**
Uni & Bigrams (TF-IDF)	449987 (all)	0.891	0.847	0.869	0.854	0.897	0.875	0.873	0.872	0.872
Uni & Bigrams (TF-IDF)	8192 (optimal)	0.960	0.890	0.924	0.897	0.963	0.929	0.929	0.926	0.926

"third_party" has higher TF scores, on average, for the privacy class, whereas the TF scores for the unigrams "third" and "party" are about the same for the two classes.

Third, the performance of the classifier increases as the number of features increases, initially. However, after a certain threshold, further increasing the number of features reduces both precision and recall. We conjecture that models trained with more features than this threshold overfit the training data. Thus, as Table 2.6 demonstrates, choosing an optimal number of features yields a classifier with better performance than the one choosing all features yields.

2.5.2 Classification

We identified that choosing unigrams and bigrams as features, TF scores as their values, and selecting a subset of features yields the best classifier results for our dataset. Considering this combination, Table 2.7 compares performances of the three machine-learned classifiers (Naive Bayes, SVM, and Random Forests) and the three baselines we evaluate. We observe the following from these results.

First, we find that machine-learned classifiers based on our feature engineering perform better than keyword-based baselines. Specifically, the SVM-based classifier performs best, overall, with the mean F_1 score of 93.1%, which is about 12% higher than the best F_1 score among the three baselines. This suggests that systematically building an incident classifier (data curation, feature engineering, and classification) is worth the effort.

Second, we note that even with an optimal number of features, the datasets we train the machine-learned classifiers on consist of considerably more features than the number of data instances. SVM is known to work-well in high-dimensional spaces. We conjecture

Table 2.7: Performances for different classification techniques (having 8192 selected TF unigrams as features) and baselines.

	Privacy Class			Non-privacy Class			Mean		
Classifier	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
Naive Bayes	0.863	0.924	0.892	0.919	0.853	0.884	0.891	0.889	0.888
SVM	**0.962**	0.897	**0.929**	**0.903**	**0.965**	**0.933**	**0.933**	**0.931**	**0.931**
Random Forests	0.931	0.820	0.872	0.839	0.939	0.886	0.885	0.879	0.879
Privacy keyword	0.918	0.700	0.794	0.757	0.937	0.838	0.838	0.819	0.816
Privacy & Solove keywords	0.664	0.930	0.775	0.783	0.530	0.663	0.774	0.730	0.719
Top-3 keywords	0.721	**0.932**	0.813	0.904	0.639	0.749	0.812	0.785	0.781

this to be a reason why SVM performs better than Naive Bayes and Random Forests in our setting.

Third, we observe that employing "privacy" as the keyword yields a classifier with a high precision (91.8%) on the privacy class. This indicates most articles mentioning "privacy" tend to be related to a privacy incident. However, we note the recall of this classifier for the privacy class is only 70%. Thus, many articles that are about a privacy incident do not explicitly mention the word "privacy."

Fourth, we observe the keyword-based classifiers based on Privacy and Solove keywords, and Top-3 keywords both yield a higher recall on the privacy class compared to the Privacy keyword-based classifier. That is, whereas searching only for "privacy" recalls only 70% of articles about privacy incidents, adding more keywords recalls more than 90% articles about privacy incidents (notice from Table 2.5 that privacy is one of the top-3 keywords). However, it is important to note that, in these cases, as the recall improves precision reduces (i.e., false positives increase), consistently. Figure 2.4 demonstrates this trend of increased recall and reduced precision, as K increases in the Top-K keyword-based classifier.

2.6 Discussion and Conclusion

We have developed an incident classifier in support of a privacy incidents database to inform research, practice, and policy making. The classifier recognizes information about privacy incidents from news articles. Since finding privacy-related information online is like searching for a needle in a haystack, the classifier can be of great value in bootstrapping the incidents database and sustaining it, by reducing the overall human effort required to add an incident to the database.

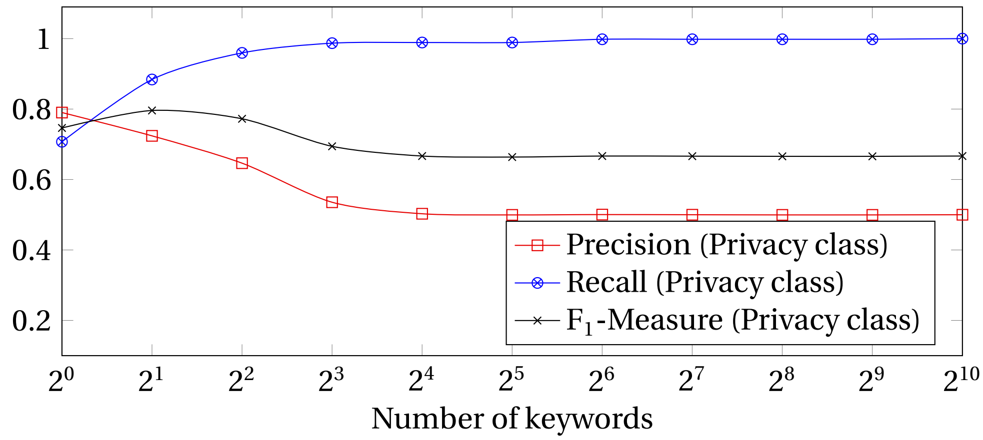

Figure 2.4: Performances of keyword-based classifiers trained on different top-K keywords.

Our evaluation suggests that machine-learned incident classifiers outperform keyword search based approaches in recognizing news articles about privacy incidents. The main challenge with keyword based approaches is that identifying a good set of keywords is nontrivial. Although we sought to systematically develop keyword sets, the sets have limitations. For example, we added terms from the Solove categories and subcategories (Solove January 2006), literally, as keywords. However, an article about "secondary use" (a Solove subcategory) may not explicitly contain the phrase "secondary use." Similarly, considering Top-K keywords may yield keywords that are in both privacy and nonprivacy articles (e.g., "Google," "Facebook," and "Apple" in Table 2.5, and are thus, not effective in classifying news articles. Nonetheless, the keyword based approaches achieve a high recall, even better than the machine-learned classifiers. As Figure 2.4 shows, with as few as Top-8 keywords, the corresponding classifier's recall on the privacy class is about 99%. Since an expert reviews articles tagged by the classifier before adding them to the database, high recall is desirable. However, since the precision of these classifiers is quite low (e.g., the precision of the Top-8 keyword based classifier in the privacy class is only 53.5%). Precisely measuring the tradeoff between increasing recall and reducing human effort (lower precision means more false positives and more reviewing work for experts) is an open problem.

We have deliberately trained our classifiers on a large set of positive (i.e. privacy incident-related) articles, rather than on training data that is representatively balanced with positive and negative. We did this both because a representative set would have few if any positive examples, given the low support of privacy incidents, and because, as the complexity of privacy taxonomies (Solove January 2006; Koops et al. 2016) demonstrates, privacy incidents are varied in nature and so argue for more training data. That said, using a large positive training set has the potential of making the classifier's job harder as well, particularly, because we included many "grey" examples (i.e., incidents that concern security but not privacy). Indeed, doing so may make it appear more likely that an article both has a feature and is related to a privacy incident than is actually the case.

CHAPTER

3

GENERATING EXPLORABLE NARRATIVE SPACES WITH ASP

RQ 2: *Can we design a narrative planner that supports:*

1. *Encoding of policies as narrative constraints*
2. *Encoding of arbitrary rulesets*

3.1 Introduction

To incorporate the constraints of complex rule systems and to support the varied modes of interaction envisioned, we need a narrative generation system that is more flexible and extensible than those available today. Past approaches have required developing a new planner implementation for each set of constraints deemed relevant to the narrative domain (such as intention in IPOCL (Riedl and Young 2010) and conflict in Glaive (Ware and Young 2011)). Further, they support primarily one mode of interaction: given a planning domain (set of actions) and problem (initial and goal scenario), generate a plan to reach the goal from the initial state. The more general framework we propose supports multiple interaction modes, including partial domain specification (to be filled in by the system), querying narrative spaces (e.g. "How many stories are there where a certain event occurs?"), and other modes of interactive exploration and iterative refinement of narrative models.

To address these goals, we present a narrative planning engine implemented with Answer Set Programming (ASP), and show how to layer narrative constraints and formulate use-specific interactions expressed as ASP clauses. We use the expressiveness afforded by ASP constraints to implement notions of intention, conflict, and belief proposed as key aspects of character believability by previous work (Riedl and Young 2010; Ware and Young 2011; Wadsley and Ryan 2013; Eger and Martens 2017; Shirvani et al. 2018). We then show how global narrative constraints, such as story grammars and plot structures, can be incorporated. Finally, we present a range of modes of interaction with the system by showing how to introduce scenario-specific constraints and ask a wide range of questions about the encoded narrative possibility space. The particular mode of interaction determines whether the system generates narrative, returns an answer to a specific query, or a combination of both.

Our key claim is that the proposed model generalizes prior narrative generation models and has the potential to enable several new, compelling modes of interaction with a narrative space. The contributions detailed in this paper to support this claim are: 1) our ASP implementation in Clingo (Gebser et al. 2008) of an intentional narrative planner, enabling the use of expressive narrative constraints; 2) a methodology to encode arbitrary constraints that interact with the narrative possibility space, using the BRUTUS model of narrative themes like betrayal (Bringsjord and Ferrucci 1999) as a case study; and 3) example encodings of question-and-answer interactions, demonstrating the versatility of our approach. Our long-term aspiration is to use this constraint-based narrative generation system as the back-end for a user-facing explorable, interactive system that will deepen user understanding of complex rule systems.

3.2 Related Work

The idea of using theorem proving technology to carry out narrative generation is not new. The BRUTUS system (Bringsjord and Ferrucci 1999) positions itself in this way: "we approach story generation through logic; in more specific terms, this means that we conceive of story generation as theorem proving" (Bringsjord and Ferrucci 1999). Mueller (2007) present a computational model of narrative that combines model finding and planning; however, their goal is story understanding rather than generation. A more recent line of work maps story domains into propositions in linear logic (Martens et al. 2013, 2014; Bosser et al. 2011), which allows for logical proofs to be directly mapped to stories; however, this approach depends on a specialized logic that is less readily adapted to meta-analysis and constraint within a standard logic programming language.

Outisde of the narrative domain, we are motivated by prior work in what we describe generally as systems that enable *explorable formal models*. Semantic modeling tools such as Rosette Torlak and Bodik (2013), Alloy Jackson (2012), PLT Redex Klein et al. (2012), the K Roşu and Şerbănuţă (2010) semantics language, the Spoofax language workbench Kats and Visser (2010), Razor Saghafi and Dougherty (2014), and miniKanren Byrd (2009) provide a basis for authoring formal specifications and programs that manipulate them, usually centered on applications to programming language theory and design. These tools support interactive execution and model querying with the same versatility that we aim to provide for the narrative domain.

Our work is probably most closely related to the RoleModel system (Chen et al. 2010), whose efforts to develop a narrative generator in ASP via the Event Calculus inspire our own approach. However, RoleModel is explicitly *not* a planning-based approach, lacking mechanisms for modeling character intentions, causality, or alternative timelines. RoleModel also limits the constraints considered relevant to narrative generation to forbidding and requiring certain roles and actions.

3.3 An Answer Set Programming Approach to Narrative Planning

The use of planning for narrative generation has a long history (Young 1999; Porteous et al. 2010; Riedl and Young 2010; Ware and Young 2011). In this approach, narratives are represented as causally-linked events, each of which changes the facts that are true in the world model. Two key attributes are considered important for the believability of a narrative: the *causal progression of plot* and *character believability*. To ensure believability of characters with cognitive processes, they must be *intentional agents*. In other words, each action a character takes should be in service to a goal (Riedl and Young 2010). A character's intentions are established as part of the narrative itself, and then drive their actions. This lets us explain a character's actions and generate examples which do not seem contrived.

Our software system consists of a general-purpose narrative planner supporting intention and conflict, implemented using Answer Set Programming (ASP). ASP is a declarative programming technique where programs are sets of logical expressions (i.e. *logic programs*), specified in terms of generative rules, facts, and constraints. A constraint solver then attempts to find stable models (answer sets) for the logic program (Gelfond and Lifschitz 1988). As opposed to other logic programming languages like Prolog, ASP provides a means of expressing disjunctive clauses through *choice rules*, which allow multiple possible worlds to be consistent with a given program. Specifically, we used Clingo (Gebser et al. 2008) as our implementation language. Lifschitz (2002) also present an ASP-based approach to narrative generation. However, due to a linear representation of the timeline, their approach cannot be used to support additional modes of interaction which require representation of causal connections.

Implementing a planner in ASP opens up the possibility of using the rich constraints provided by the system for sculpting the possibility space of generated narratives. The interplay between generative rules and constraints, coupled with relational programming, allows a model, once specified, to be used in several different ways. An ASP model merely specifies the relationships between different predicates. Additional constraints can be applied later to use some of the predicates as "inputs" to the program, while the solver finds consistent values for the rest.

3.3.1 Preliminaries

ASP can be used to generate narrative scenarios from a possibility space of action sequences, as explored initially in the RoleModel system by Shen et al. (Chen et al. 2010). Possibility spaces are represented through *choice rules* of the form $\{\phi\}$, which indicate that the formula ϕ may, but is not required to, hold in each model. For example, the following program expresses the possibility for a user to do any, both, or neither of two post actions:

```
{happens(attack(princess, robot)); happens(attack(robot,
   princess))}.
```

A standard answer set solver will report these results as:

```
Answer: 1

Answer: 2
```

```
happens(attack(robot,princess))
Answer: 3
happens(attack(princess,robot))
Answer: 4
happens(attack(robot,princess)) happens(attack(princess,robot)
  )
```

Logic variables (indicated syntactically as identifiers starting with a capital letter) and conditions (formulas to the right of the backwards implication symbol `:-`) can be used to quantify over finite sets and generate possible clauses for each element. For example, the following rule generates a set of satisfying models in which `happens` may or may not hold for each action specified as "possible:"

```
{ happens(Action) } :- possible(Action).
```

The resulting generative space has size $2^{|Action|}$.

3.3.2 Event Calculus

In narratives, the state of the world changes over time. It is common to formalize state change with predicates whose truth depends on a temporal index; these time-varying predicates are called *fluents*. *Event calculus* (Kowalski and Sergot 1989) is a particular logical theory of fluents used to represent and reason about sequences of events. The state evolves through *actions*, which describe how fluents may change from one time step to the next. They may require preconditions to be satisfied before they can be executed. Actions are assumed to be instantaneous, and occur in discrete time steps. A common formalization of event calculus axioms is as follows:

$$holds(T+1,F) \leftarrow initiates(T,F) \tag{3.1}$$

$$holds(T+1,F) \leftarrow holds(T,F) \wedge \neg terminates(T,F) \tag{3.2}$$

Part (3.1) states that if an action at time T $initiates$ (makes true) a fluent, then the fluent $holds$ (is true) at the next time step. Part (3.2) states that fluents have inertia, i.e., they continue to be true until $terminated$ by an action. This assumes default negation (if predicate P cannot be derived, then $\neg P$ is assumed to hold). This representation can be readily translated into ASP.

3.3.3 Event Calculus for Partially Ordered Events

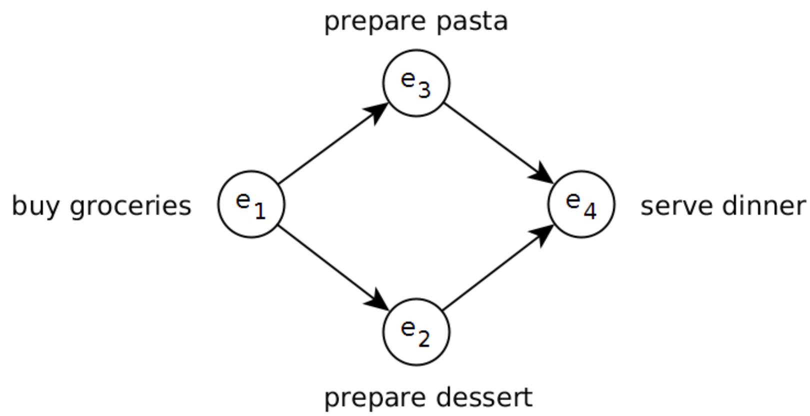

Figure 3.1: Consistent partial order

In narratology, there are two aspects to a narrative: *fabula* (the chronological order of events), and *syuzhet* (the order in which events are narrated). Generating an interesting syuzhet requires considerations that are beyond the scope of this paper. We will limit ourselves to generating the fabula, which suits our goal of making complex rule sets explorable. We define a fabula as a sequence of causally linked events (actions). Actions naturally form a partial order because of their prerequisites and effects. Some actions must be performed before others, while others may be mutually independent. For instance, in the sequence of events *buy groceries* → *prepare pasta* → *prepare dessert* → *serve dinner*, we can see that *buy groceries* must occur first, *serve dinner* must occur last, but the other two events can occur in either order in between them. We can depict this as a diamond graph (Figure 4.2), where e_2 and e_3 are *unordered*. This is a better representation than a linear graph, since it allows us to represent causality and to arbitrarily reorder independent parts of the graph.

The standard event calculus formulation works well for a totally ordered sequence of events, but breaks down if we have a partial order. Consider the four events shown as the nodes of a graph in Figure 3.2. Here we can be sure that $holds(e_4, P_1)$ is true, but we cannot say anything about $holds(e_4, P_2)$. In fact, (3.1) and (3.2) would incorrectly imply that $holds(e_4, P_2)$ is true.

We therefore extend the standard formulation to allow for partially ordered events by introducing a new predicate, $strong\_holds$, which we define as follows:

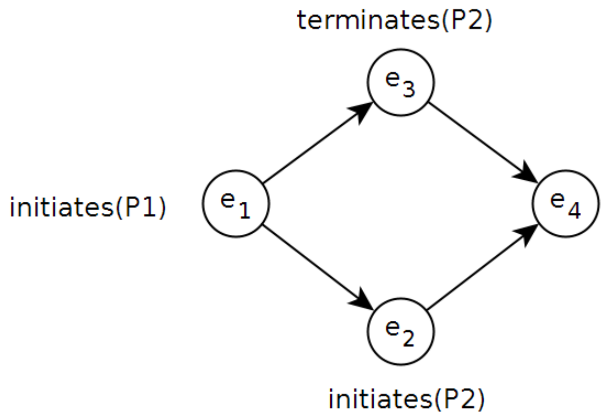

Figure 3.2: An example of an ambiguous partial order. The arrows represent the partial order relation *happens before*. This relation is transitive, but we have omitted the edge between e_1 and e_4, since it can be derived through transitivity. We will omit such edges throughout the paper for clarity. Nodes are labeled with the effects of each action.

$$strong\_holds_{\mathscr{P}}(e,F) \iff \forall \mathscr{T} \in \boldsymbol{C}(\mathscr{P})\, holds_{\mathscr{T}}(e,F) \tag{3.3}$$

The predicate subscripts indicate the order over events in which the predicate holds, and $\boldsymbol{C}(\mathscr{P})$ is the set of all total orders consistent with the partial order $\mathscr{P}$. We say that total order $(A, <_{\mathscr{T}})$ is consistent with partial order $(A, <_{\mathscr{P}})$ iff:

$$\forall x, y \in A(x <_{\mathscr{P}} y \implies x <_{\mathscr{T}} y)$$

In other words, $strong\_holds(e,F)$ is only true if we can be sure that the fluent will hold regardless of how we “linearize" the partial order. Going back to figure 3.2, we can conclude that $strong\_holds(e_4, P_1)$ is true, but not $strong\_holds(e_4, P_2)$.

This, however, creates the issue that we can no longer rely on default negation. For instance, while $strong\_holds(e4, P_2)$ is not true, we also cannot say for sure that the fluent P_2 will not hold at e_4. So we introduce another predicate, $strong\_notholds$, which is defined in a similar manner:

$$strong\_notholds_{\mathscr{P}}(e,F) \iff \forall \mathscr{T} \in \boldsymbol{C}(\mathscr{P}) \neg holds_{\mathscr{T}}(e,F) \tag{3.4}$$

where all other terms are as defined in (3.3).[1] Then we can force the planner to generate se-

[1]For implementation in Clingo, we need to translate this statement to an equivalent formalization express-

Figure 3.3: The definition of an intention frame, depicted schematically.

quences where the partial order matches causality by maximizing the number of unordered nodes in the graph,[2] as shown in Listing 3.1.

Listing 3.1: Showing causal links by maximizing the number of unordered nodes

```
unordered(Node1, Node2) :-
    id(Node1), id(Node2),
    not after(Node1, Node2),
    not before(Node1, Node2),
    Node1!=Node2,
    concludes(C, Node1):
        terminus(C),
        concludes(C, Node2). % Same timeline
#maximize
    {1@1, unordered(Node1, Node2): unordered(Node1, Node2)}.
```

3.4 Narrative Constraints

3.4.1 Intention

Character intentionality refers to the idea that each action a character takes in the generated narrative should be in service to a goal (Riedl and Young 2010). A character's intentions are established as part of the narrative itself, and then drive their actions. We adopted this idea in our planner because it lets us explain a character's actions and generate examples which do not seem contrived.

Figure 3.3 presents a schematic representation of a key part of our logical definition of intention. An intention created at time T_1 is *satisfied* at some later point T_2 iff T_2 initiates the

ible in Clingo's language of Horn clauses. We omit the expanded definition for brevity.

[2]Clingo supports maximization of the cardinality of a given set through the `#maximize` directive.

intended event, and there is no earlier timepoint T' between T_1 and T_2 which satisfies it. We then define an intention frame by saying that any timepoint between when the intention is formed and satisfied, on which the satisfying event is causally dependent, is within the frame. Our ASP implementation is shown in Listing 3.2.

Listing 3.2: Definition of intention frames and intentionality of plans.

```
% N1, N2, N3: graph nodes
% F: intended fluent
% A: actor
%% Defining satisfies(N, I)
satisfies(N2, intention(N1, F, A)) :-
    intends(N1, F, A), initiates(N2, _, F),
    after(N2, N1),
    not satisfies(N3, intention(N1, F, A)) :
        after(N3, N1), before(N3, N2).

% CONSTRAINT: All intended fluents must hold
% at some point after they're intended
:- intends(N1, F, A),
    not satisfies(_, intention(N1, F, A)).

%% Defining frame(N, I)
frame(N3, intention(N1, F, A)) :-
    satisfies(N2, intention(N1, F, A)),
    after(N3, N1), before(N3, N2),
    actor(N3, A).

% CONSTRAINT: All events must be part of
% frames of all their actors
:- actor(N, A),
    not frame(N, intention(_, _, A)),
    not satisfies(N, intention(_, _, A)),
    A!=env.
```

3.4.2 Conflict

Conflict is a central concept of European-originating narrative theory: a story is generally considered interesting only if different characters (intentional agents, which may include nonhuman entities such as the environment or society) act towards conflicting goals (Ross 1993). The CPOCL narrative planner models conflict as a character's plans being thwarted by another character (Ware and Young 2011). In other words, a conflict occurs when two or more intentions are mutually incompatible. We incorporate this model into our planner, allowing us to model real-world scenarios with antagonistic actors in addition to fictional scenarios with greater narrative interest.

The CPOCL model induces a notion of alternate, or branching, timelines, in the sense that for a plan to be thwarted, that plan has to have been intended to take place by the thwarted agent. A given timestep can thus have multiple *conclusions*. We define a conclusion of a given timestep as any terminus that occurs after it. We then define a *split* in the timeline, indicating unexecuted actions, as occurring when a child node does not have all of the conclusions of its parent. Finally, a node where timeline splits should have a single parent, in order to avoid scenarios where a timeline feeds back into another timeline. This is shown in Listing 3.3.

Listing 3.3: Definition of conflict creating split timelines.

```
terminus(C) :-
    id(C), not after(N, C): id(N).
concludes(C, N) :-
    id(N), after(C, N), terminus(C).
concludes(C, C) :- terminus(C).

split(N1, N2) :-
    edge(N1, N2),
    concludes(C, N1),
    not concludes(C, N2),
    C!=N2.
:- split(N1, N2),
    not { edge(P, N2): id(P) } = 1.
```

3.4.3 Plot Schema

BRUTUS (Bringsjord and Ferrucci 1999) is a narrative generation engine that encodes themes such as *betrayal* and generates stories that contain these themes. For example, BRUTUS uses a Prolog-like language to "mathematize" betrayal as shown in Listing 3.4.

Listing 3.4: BRUTUS plot schema definition for betrayal.

```
betrayal(Betrayor, Betrayed) :-
    goal(Evil, EvilPlan, Betrayor),
    includes(EvilPlan, BetrayorsLie),
    say(BetrayorsLie),
    includes(EvilPlan, Thwarting),
    thwart(Thwart),
    prevented_goal(Thwarting, BetrayedsGoal),
    supports(BetrayorsLie, BetrayedsGoal),
    goal(BetrayedsGoal, BetrayedsPlan,
      Betrayed),
    belief(Betrayed, BetrayorsLie).
```

The concept of themes of this nature, which we refer to as *plot schema*, dates back to pre-digital notions of narrative grammars, as in Propp's foundational *Morphology of the Folktale* (Propp 1928). Narrative grammars informed early computational approaches to narrative formalization, and they have an ongoing legacy in more recent work such as *plot units* (Lehnert 1981; Goyal et al. 2013) and *story intention graphs* (Rishes et al. 2013). While planning-based approaches afford much richer world models, narrative grammars and plot schema remain useful as a way of codifying *global structure* to plots, allowing us to identify common plot structure between distinct story contexts (e.g. Disney's *The Lion King* as an adaptation of Shakespeare's *Hamlet*). This model is also useful if, say, we want to require generated narratives to carry particular themes or tropes, follow a particular narrative arc (pattern of rising and falling action), or subvert genre expectations for any of the above.

We show how our planner can gracefully account for plot schema like this by encoding the betrayal example into the notions of intention and conflict supported by our planner (presented here in natural language for readability):

- The Betrayed has intention I_1.
- The Betrayer expresses intention I_2, but has intention I_3.

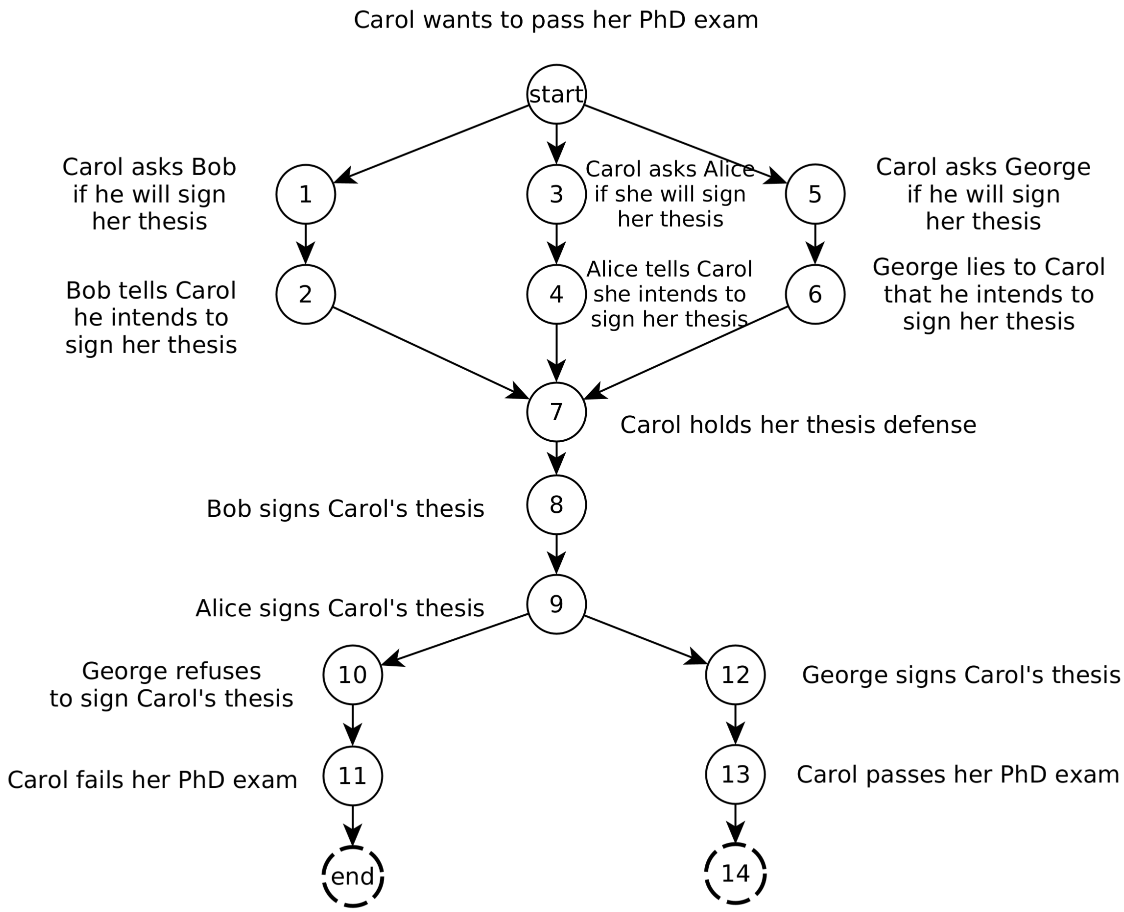

Figure 3.4: A betrayal narrative generated by our system.

- Step S_2, which would satisfy the expressed intention I_2, lies inside the intention frame of the Betrayed's intention I_1.
- Step S_2 occurs in an alternate timeline, in which I_1 is also satisfied.
- Step S_3 occurs on the actual timeline and satisfies I_3.

Figure 3.4 shows a betrayal narrative generated by our system using this encoding, using an example domain inspired by BRUTUS. The Betrayed (Carol) needs 3 signatures on her thesis to pass her PhD exam (I_1). The Betrayer (George) promises to sign her thesis (I_2), but instead has the intention to see her fail (I_3). The main timeline extends from the $start$ node to the end node, and satisfies the Betrayer's intention. The Betrayed's intention is satisfied in an alternate timeline (extending from $start$ to node 14), and depends on the Betrayer's promise.

3.5 Example Interaction

Our central claim is that our ASP formulation unifies and generalizes prior work on plan-based narrative generation, supporting several modes of interaction through slight modifications on input constraints. For instance, instead of just generating narratives from a domain, a user could use additional constraints to gain an understanding of the underlying rule set of the domain. This could be useful in making complex realworld rule sets, e.g. laws, more accessible through explorable examples. This section validates the final claim by demonstrating an example domain and interaction.

Consider a domain where a dragon needs to be slayed. There are various ways to achieve that goal, but the dragon has a specific weakness. The goal of the user is to find that weakness.

Our hero may ask one or more of: Ice Wizard, Air Bender, or Fire Demon to accompany them.
They may take one or more of: Earth Stone, Fire Lantern, or Ice Diamond with them.
They may travel through one or more of: Fire Pit, Air Gardens, or Metal Caves.

To find the weakness, the user first asks the system to generate a few narratives where the dragon is successfully slayed.

Listing 3.5: Generating successful narratives

```
:- not strong_holds(end, dragon_slayed).
```

The user sees the following two narratives:

Narrative 1:

- The hero asks the Ice Wizard to accompany them
- The hero travels through the Fire Pit
- The hero acquires the Earth Stone

Narrative 2:

- The hero asks the Fire Demon to accompany them
- The hero travels through the Air Gardens
- The hero acquires the Earth Stone

The user observes two common elements: earth stone, and fire. The user wants to check whether a successful narrative can be generated without using either.

Listing 3.6: Narrowing down

```
:- not strong_notholds(end, acquired(earth_stone)).
:- type(N, fire), id(N).
```

The system then generates the following narrative, which means indeed, it is possible:

- The hero asks the Air Bender to accompany them
- The hero acquires the Ice diamond
- The hero travels through the Metal Caves

The user notices that the sequence always contains 3 distinct element types. At this point, the user would like to know what an unsuccessful narrative looks like.

Listing 3.7: Unsuccessful narrative

```
:- not strong_notholds(end, dragon_slayed).
```

Narrative 1:

- The hero asks the Air Bender to accompany them
- The hero acquires the Ice diamond
- The hero travels through the Air Gardens

Narrative 2:

- The hero asks the Air Bender to accompany them
- The hero travels through the Metal Caves
- The hero acquires the Earth Stone
- The hero travels through the Air Gardens

At this point, it seems that the occurrence of the same element more than once prevents the dragon from being slayed, and we require 3 distinct elements. To confirm this, the user asks the system if an unsuccessful narrative is possible when these conditions are met:

Listing 3.8: Unsuccessful narrative

```
:- 2 { N,T: type(N, T)}, type(T).
:- { T: type(N, T) } 3.
:- not strong_notholds(end, dragon_slayed).
```

The system returns `UNSATISFIABLE`, which means the dragon will always be slayed with these conditions.

3.6 Discussion

There are some limitations of our work that need to be addressed. First, while we have a notion of causality and ordering of events, the narratives do not specify how this ordering relates to time in the real world in days and hours. This can be relevant when encoding constraints that mention specific time intervals.

Second, we did not focus much on optimization, and state space explosion can create performance issues. For narratives longer than 20 steps, grounding can result in a 1GB+ file, with a memory usage of 8GB+. We will focus on optimization in the next phase of our work. We are also looking at recent research in lazy grounding (Taupe et al. 2017).

Third, when generating all possible answer sets, many of the narratives generated are very similar and differ only in inconsequential details, like graph node identifiers. The user can usually apply additional constraints to narrow down to the desired narrative, but this is less than ideal.

While Clingo proved to be a good choice for representing our system, we did face some hurdles during implementation. A lack of support for typed predicate arguments means that typos in predicate names and arguments often went unnoticed. This issue could be alleviated by an extension to the language allowing explicit typing and static type checking.

3.7 Conclusion

This work unifies several previous narrative models within a relational programming framework that can be adapted to a number of use cases. We presented a flexible, CPOCL-style narrative planner written in pure ASP, showing how prior work on intention and conflict could be represented elegantly. We then demonstrated the versatility of this approach by incorporating thematic constraints via *plot schema* and showing how a rich interaction could be supported with small changes to the input.

CHAPTER

4

POLICIES AS PLANNER CONSTRAINTS

RQ 2: *How do we encode the privacy-specific knowledge contained in real-world policies and regulations like HIPAA in the context of a narrative planner?*

4.1 Introduction

Today, our private information is distributed across various entities, from social media apps to healthcare providers. As the systems that store this information become more complex, so do the rules and laws governing them. From a user perspective, an understanding of these policies is crucial to making an informed judgement when giving away personal information. Despite this, research shows that even well-educated users have trouble understanding these policies Proctor et al. (2008). This lack of understanding can create an *information asymmetry* between providers of these services and the user, where the user is not fully aware of the consequences of their actions, or even of when an entity has gained access to their information Acquisti and Grossklags (2007). This lack of understanding has been proposed as a possible explanation to the “privacy paradox" – the disparity between people's attitudes and behaviours toward privacy Kokolakis (2017).

On the other side of the equation, the increasing ease of data collection and aggregation enabled by technology also makes policy design a difficult task, as it becomes harder to anticipate the consequences of allowing or disallowing a particular form of collection. As Adams and Sasse (2001) state, "Most invasions of privacy are not intentional but due to designers inability to anticipate how this data could be used, by whom, and how this might affect users."

Existing work to solve these issues has focused on one or the other. Efforts addressing the user comprehension problem have focused on better presenting the policies to users Cranor (2003); Lin et al. (2012); Cranor et al. (2006), while ignoring the policy design side of things. Ultimately, the failure of efforts such as P3P was because of a lack of adoption by information workers Cranor (2012). On the other hand, research to enable policy verification has focused on formalization of policies and provable guarantees Chowdhury et al. (2013); DeYoung et al. (2010), not addressing the misalignment between policy meaning and user understanding Reidenberg et al. (2015).

We argue that an approach combining formal policy modeling techniques with narrative generation methods can help solve both these issues. This approach seems viable for a few reasons.

First, while policies and regulations are expressed as a set of complex rules, humans often like to learn by generalizing from a series of examples. Narratives depicting plausible real-world scenarios that follow these rules would naturally serve as examples to educate users, while also addressing the context-dependence of user perceptions of privacy Nissenbaum (2004). Fortunately, the structured language of most regulations tends to be a good fit for formalization, as demonstrated by existing law formalization efforts. As we will show in this paper, these formalizations can also be adapted to work in a generative system.

Second, a constraint-based narrative generation system makes policies explorable and interactive, allowing a user to query the system for the particular aspect of the policy relevant to them, by applying constraints. We expect this to be far more accessible than the often terse and formulaic legalese regulations are written in.

Third, from a policy design point of view, our system can be used to hunt down edge cases and do sanity checks, while testing the effect of any changes on various scenarios.

The goal of this work is not to create another policy modeling logic. It fundamentally differs from the existing efforts, in that it is a *generative* system, as opposed to a *compliance verification* system. While it can be used to verify compliance of a given trace of actions (it does so by attempting to generate a matching narrative), its strength lies in enabling exploration of the encoded regulation through scenarios involving intentional actors.

4.2 Related Work

In this section, we describe prior work in the related areas of policy formalization and policy explanation.

4.2.1 Policy formalization

The increasing complexity of regulations makes it desirable to have an automated reasoning system capable of verifying compliance. While legal text often contains ambiguities and cross-references Bench-Capon et al. (1987), its structured language lends itself readily to a logical representation. To this end, various modeling approaches have been used, including defeasible logic to handle overlaps and conflicts in legal text Antoniou et al. (1999); Governatori et al. (2009) and deontic logic to represent permissions and obligations Valente and Breuker (1995). For a survey of other related work in this area, we refer the reader to Otto and Antón (2007).

We now look at some efforts specifically targeting privacy laws.

Barth et al. (2006, 2007) present LPU (Logic of Privacy and Utility), a formalization of "Contextual Integrity", applied in the context of business processes to ensure privacy. DeYoung et al. (2010) build on that work with PrivacyLFP logic. Their logic supports obligations and temporal reasoning. They use it to produce a comprehensive formalization of transmission-related HIPAA clauses. Lam et al. (2009) describe a system for verifying HIPAA compliance of messages sent by hospital employees. They present pLogic, an executable framework implemented as a Datalog program. Their formalization does not support ensuring future obligations. Chowdhury et al. (2013) present a policy specification language based on first-order linear temporal logic, along with an algorithm for static policy analysis of policies like HIPAA.

While our work adopts some ideas from the above, as a generative system, it remains fundamentally different.

4.2.2 Policy explanation

Work in this area focuses on presenting privacy policies to users in a more comprehensible way. P3P Cranor (2003) was a system for allowing websites to declare the intended usage of collected information. The browser could then block any cookies that would conflict with user privacy settings. Cranor et al. (2006) present a better user interface for a P3P agent. Lin et al. (2012) use crowdsourcing to capture user's expectations of privacy in the context of mobile apps. They describe a privacy summary interface that emphasizes places where these expectations are broken. Kelley et al. (2009) present a "nutrition label" for privacy, displaying in a grid what categories of information are accessed to what extent, drawing from the FDA's nutrition facts panel. Finally, Apple now shows privacy labels in its App Store listings, based on a questionnaire filled in by the developer regarding what information their app collects and how it is used.

These efforts focus on presenting static information to the user in a usable manner. Our system enables a richer exploration of regulations through intentional narratives.

4.3 System Overview

Our system consists of four main components, as shown in Figure 4.1.

4.3.0.1 Clingo

Clingo is an Answer Set Programming (ASP) system Gebser et al. (2008). ASP is a declarative programming technique where logic programs are specified in terms of generative rules, facts, and constraints. A constraint solver then attempts to find stable models (answer sets) for the logic program Gelfond and Lifschitz (1988). As opposed to other logic programming languages like Prolog, ASP allows specifications of "choice rules", which allow multiple possible worlds to be consistent with a given program. This is well-suited to generative applications like ours.

Using ASP instead of an imperative language has a couple of advantages: 1) it allows straightforward representation of logical rules, and 2) the interplay between generative rules and constraints, coupled with relational programming, allows a model, once specified, to be used in various ways. An ASP model merely specifies the relationships between different predicates. Additional constraints can be applied later to use some of the predicates as "inputs" to the program, while the solver finds consistent values for the rest.

4.3.0.2 Narrative planner

A narrative ("story") is a sequence of causally linked events (actions). Actions naturally form a partial order because of their prerequisites and effects. Some actions must be performed

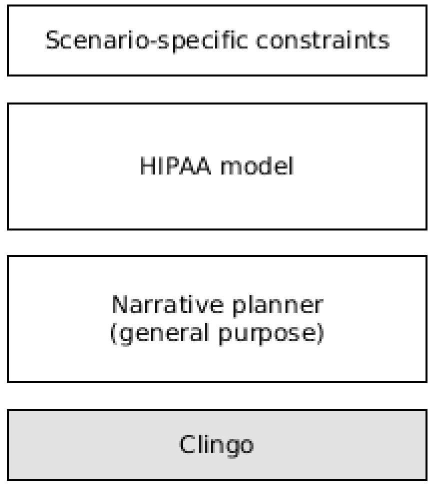

Figure 4.1: System overview

before others, while others may be mutually independent. For instance, in the sequence of events *buy groceries* → *prepare pasta* → *prepare dessert* → *serve dinner*, we can see that *buy groceries* must occur first, *serve dinner* must occur last, but the other two events can occur in either order in between them. We can depict this as a diamond graph (Figure 4.2), where e_2 and e_3 are *unordered*. This is a better representation than a linear graph, since it allows us to trace causality, an important feature for verifying policy compliance.

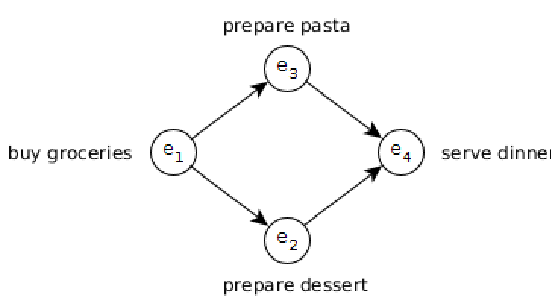

Figure 4.2: Consistent partial order

A narrative planner generates this sequence of causally related events, based on initial conditions, available actions, and any additional constraints. Implementing a planner in ASP opens up the possibility of using the rich constraints provided by the system for sculpting the possibility space of generated narratives using privacy policies.

4.3.0.3 HIPAA model

We use the expressiveness afforded by ASP constraints to implement a partial model of the Health Insurance Portability and Accountability Act (HIPAA) transmission rules. The model acts as a set of constraints which can be used to restrict the generated narratives to ones that comply with the regulation.

4.3.0.4 Scenario constraints

Finally, scenario-specific constraints can be added to describe a particular setting and use case. The way these constraints are specified determines whether the system generates a specific kind of narrative, verifies compliance of a given narrative, or a combination of both. We will give several examples of these in the Examples section.

4.4 Policies as Narrative Constraints

4.4.1 Overview of HIPAA

The Health Insurance Portability and Accountability Act is a US law enacted in 1996. It was created to modernize and regulate the flow of healthcare information, and protect patient privacy. In this paper, we will deal only with the part of HIPAA that regulates the transmission of information by healthcare entities.

We selected HIPAA because it is complex enough to build confidence in our approach, as well as modular enough that it's possible to encode a representative subset without having to encode the entire text. It also has potential real-world applications, since HIPAA compliance can be onerous for healthcare providers.

4.4.2 Encoding Methodology

We now describe our methodology for encoding transmission-related HIPAA clauses as narrative constraints.

4.4.2.1 Role hierarchy

Most clauses in HIPAA are conditioned on the *role* of the involved entities. For instance, §164.506(c)(5) states:

> A **covered entity** that participates in an **organized health care arrangement** may disclose protected health information about an **individual** to other **participants** in the organized health care arrangement for any health care operations activities of the organized health care arrangement.

The four roles mentioned in the clause have been bolded. The roles form a hierarchy – for instance, "covered entity" is further defined in §160.103 as either a health care provider (doctors, clinics, etc.), a health plan, or a health care clearinghouse. Therefore, a clause which applies to a covered entity must also apply to all doctors. Care must be taken to make sure that the hierarchy expresses strict "is-a" relationships. A `covered_entity` can either be a `person` (doctor) or an `organization` (hospital), so it does not come under either.

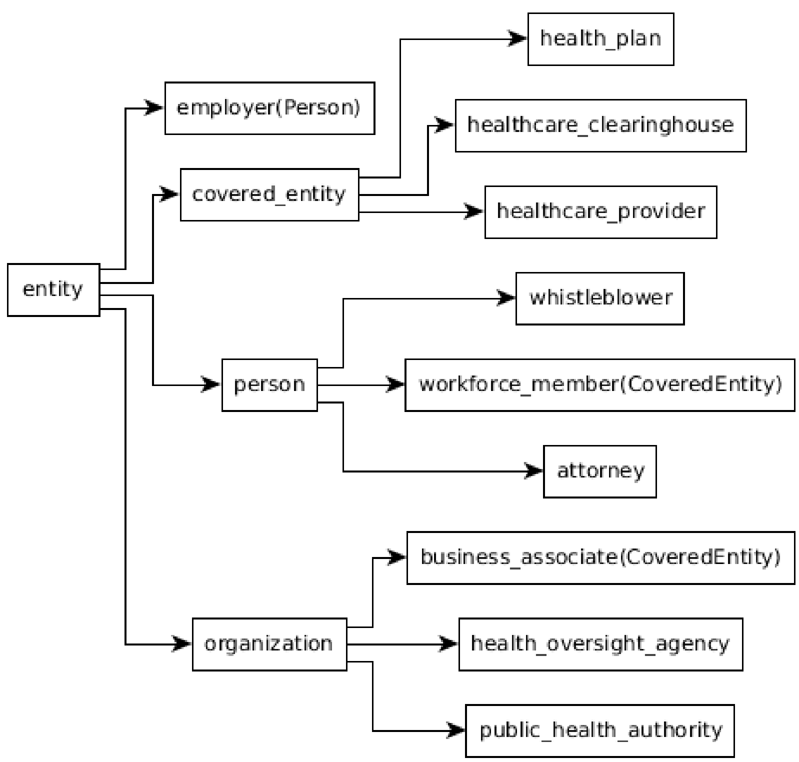

Figure 4.3: Partial role hierarchy

We show a partial graph of the hierarchy in Figure 4.3. Note that certain roles are *parameterized*. For instance, the role “participant" in the clause above refers to participation in a specific `organized_healthcare_arrangement`, the one whose health care operations are in question. These parameter types are mentioned in parentheses in the figure.

Actions in the generated narrative are carried out by *agents* with roles. An agent with a given role is also automatically assigned all parent roles in the hierarchy. Additionally, an agent may also take on an unconnected role. For instance, a doctor may also take the role of patient, with the obvious restriction that an organization cannot also be a person. These roles may be assigned manually as part of the initial conditions, or automatically by the planner in order to satisfy a given narrative constraint.

4.4.2.2 Actions

HIPAA inherently defines many `actions` that can be performed by agents. For instance, §164.512(f)(3) states:

> *Permitted disclosure: Victims of a crime.*
>
> Except for disclosures required by law as permitted by paragraph (f)(1) of this section, a covered entity may **disclose** protected health information in response to a law enforcement official’s **request** for such information about an individual who is or is **suspected** to be a victim of a crime, other than disclosures that are subject to paragraph (b) or (c) of this section, if:
>
> (i) The individual **agrees** to the disclosure; or
>
> (ii) The covered entity is unable to **obtain** the individual’s agreement because of incapacity or other emergency circumstance

Again, we have bolded the obvious actions in the clause. Each action requires: 1) an Actor, 2) pre-conditions and 3) effects. The preconditions and effects are expressed in terms of fluents (see 4.4.2.4) or the state of the world. For the purposes of HIPAA, we only need encode the preconditions that are not convered by the Transmission Conditions (see 4.4.2.6). Let’s take the example of the action **disclose**:

1. **Actor:** Any organization or person deemed to be (has a `role` of) a "covered entity" under HIPAA rules

2. **Preconditions:** The actor must possess the information being disclosed

3. **Effects:** The specific law enforcement official who made the request is now in possession of the disclosed information

4.4.2.3 Action reachability

For an action to meaningfully be part of the planning domain, it needs to be "reachable", which is to say, its preconditions must be able to be satisfied. In the above example of the action **disclose**, there is no mentioned action that can satisfy the precondition of the covered entity being in possession of said information. There are two ways of dealing with this:

1. Add the required fluents to the initial state of the world, as part of the query

2. Add "latent" actions that can satisfy the preconditions

"Latent" actions are actions that are not explicitly mentioned as part of the poilcy text, but must be encoded in order to form a coherent narrative. For instance, to *disclose* protected health information, the covered entity must first *acquire* it, and the acquired information must be about the specific patient in question. The entity might come in possession of the information through a *doctor's visit*, or any of a number of possible real world actions. We need to encode at least one such action in our model. (While the information might also be acquired through a disclosure by another entity, it must ultimately originate somewhere.) Similarly, the "incapacity" must also be the consequence of an event, or encoded as part of the query. Generally, since encoding specific fluents as part of the query requires knowledge about the specifics of the domain encoding, adding latent actions is preferable.

The given sample encoding describes the pre-conditions (`possible` predicate) and the effects (`initiates` predicate) of the action. The planner uses this information to form a coherent narrative.

Listing 4.1: Sample action encoding

```
% Giving treatment to a person is
% one way to acquire their health information
possible(I, A, al(Person, al(CE))):-
    id(I),
    A=action(get_treatment, Person, CE),
    role(Person, person),
    role(CE, covered_entity).
```

```
initiates(I, A, has_info(CE,Person,phi,I)):-
    A=action(get_treatment, Person, CE),
    happens(I, A, _).
```

4.4.2.4 Fluents

Fluents track the state of the world. They also serve to causally link actions. In §164.512(f)(3) above, the action of *agreeing* to the disclosure must occur after the attempt to *obtain* the individual's agreement is made. We track this by introducing a fluent denoting this attempt. For actions without any obvious consequences as given in the regulation text, we still maintain fluents denoting completion of those actions, so that an intentional agent can target them as goals.

Optionally, roles can also be tracked as fluents to allow them to change through the narrative. Fluents may be manually specified to be true as part of the initial condition.

4.4.2.5 Information attributes

Whether or not a HIPAA clause applies is often dependent on the type of information being transmitted. The regulation text specifies several of these types, and they also form a hierarchy. For instance, a clause applicable to "private health information" must also apply to "psychotherapy notes" (§164.508(a)(2)), since latter is a type of former.

4.4.2.6 Transmission conditions

Clauses allow or disallow transmission of protected information based on certain conditions. Additionally, they may impose future obligations on entities. Here we borrow the encoding scheme of Lam et al. (2009), with an additional term to allow future obligations to be checked.

- *Category* defines a set of conditions that say whether or not the rest of the clause is applicable.
- *Exception* refines a category's applicability.
- *Requirement* is a condition that must be met in order for the action in question to be allowed.

- *Obligation* defines a set of events that must occur in the future (in all conflicting timelines) for this action to be compliant.

$$
\begin{aligned}
permits_R \leftarrow & (category_R \wedge \neg exception_R) \\
& \wedge (requirement_R \wedge obligation_R) \\
forbids_R \leftarrow & (category_R \wedge \neg exception_R) \\
& \wedge (\neg requirement_R \vee \neg obligation_R) \\
not\_applicable_R \leftarrow & \neg category_R \vee exception_R
\end{aligned}
$$

Listings 4.2, 4.3, and 4.4 show a sample encoding of clause §164.508(a)(3):

> **Authorization required: Marketing.**
>
> (i) Notwithstanding any provision of this subpart, other than the transition provisions in § 164.532, a covered entity must obtain an authorization for any use or disclosure of protected health information for marketing, except if the communication is in the form of:
>
> (A) A face-to-face communication made by a covered entity to an individual; or
>
> (B) A promotional gift of nominal value provided by the covered entity.

Listing 4.2: Rule 164.508(a)(3)(i): category

```
rule(r164_508_a_3_i, s164_508).

msg_purpose(marketing).

category(I, r164_508_a_3_i,
    action(message, To, From,
        About, Type, Purpose)) :-
    id(I),
    action(message, To, From,
        About, Type, Purpose),
    msg_subtype(Type, phi),
    role(From, covered_entity),
    Purpose=marketing.
```

Listing 4.3: Rule 164.508(a)(3)(i): requirement

```
requirement(I, r164_508_a_3_i, A) :-
    action(message, To, From,
        About, Type, Purpose),
    A=action(message, To, From,
        About, Type, Purpose),
    strong_holds(I,
        hipaa_authorization(About, A)).
```

Listing 4.4: Rule 164.508(a)(3)(i): exceptions

```
exception(I, r164_508_a_2, A) :-
    id(I),
    action(message, To, From,
        About, Type, Purpose),
    A=action(message, To, From,
        About, Type, Purpose),
    hipaa_msg_details(I,
        A, _, _, face_to_face),
    role(From, covered_entity),
    person(To),
    To=About.

exception(I, r164_508_a_2, A) :-
    id(I),
    action(message, To, From,
        About, Type, Purpose),
    A=action(message, To, From,
        About, Type, Purpose),
    hipaa_msg_details(I,
        A, _, _, nominal_promotional_gift),
    role(From, covered_entity),
    To=About.
```

4.4.2.7 Strong and weak compliance

Barth et al. introduce the notions of *strong* and *weak compliance*. Weak compliance indicates that a proposed action, if taken at the present time, will not violate policy requirements. Strong compliance indicates that (1) the proposed action will not prevent any future obligations imposed by the policy from being fulfilled, and (2) it will allow an infinite number of weakly compliant actions to be performed Barth et al. (2006, 2007). We sidestep the issue of having to check for (1) by checking if an overall compliant narrative can be generated (using the `obligation` predicate). We are not concerned with (2), since we only deal with finite traces of pre-determined maximum length.

4.4.2.8 Tracking information flow

When a piece of health information is generated (e.g. through a patient visiting a doctor), we tag the information with the unique identifier of the action which generated it. This identifier is propagated throughout subsequent transmission of the information.

4.5 Examples

In this section we will show several examples from both user and policy design perspective, demonstrating the flexible nature of our system.

4.5.1 Policy comprehension

Here, we will explore scenarios where a user wants to understand how a given policy could apply to their particular situation. We will demonstrate how to encode each scenario and how to interpret our system's output. Our examples will be in the context of a subsection of HIPAA §164.

4.5.1.1 Example 1: Can X happen?

To begin with, we will look at how to encode a simple scenario where the user wants to know whether their hospital can sell their protected health information (PHI) to third parties.

We encode this scenario as shown in Listing 4.5.

Listing 4.5: Example 1

```
% initial conditions
role(alice, person).
```

```
role(hospital, covered_entity).
role(advertiser, entity).
unintentional(alice;hospital;advertiser).

% constraints
:- not strong_holds(end, information_sold).
```

Here we are stating that `alice` is a person, `hospital` is a covered entity as described in HIPAA §164.501, and `advertiser` is a third party that the information can be sold to. The actions that the agents can take were already encoded as part of our encoding of HIPAA, and no additional specifications are required in this case. We also state that all three agents are `unintentional`, which means that their actions are not driven by personal goals. We will see examples of intentional agents later.

We also add the constraint that at the end of the generated sequence of events, the fluent `information_sold` should be true. This is done by forbidding the negation of `information_sold`. This allows us to filter out narratives in which the condition doesn't hold, without defining the predicate to be true (i.e., the predicate must arise naturally from other parts of the program

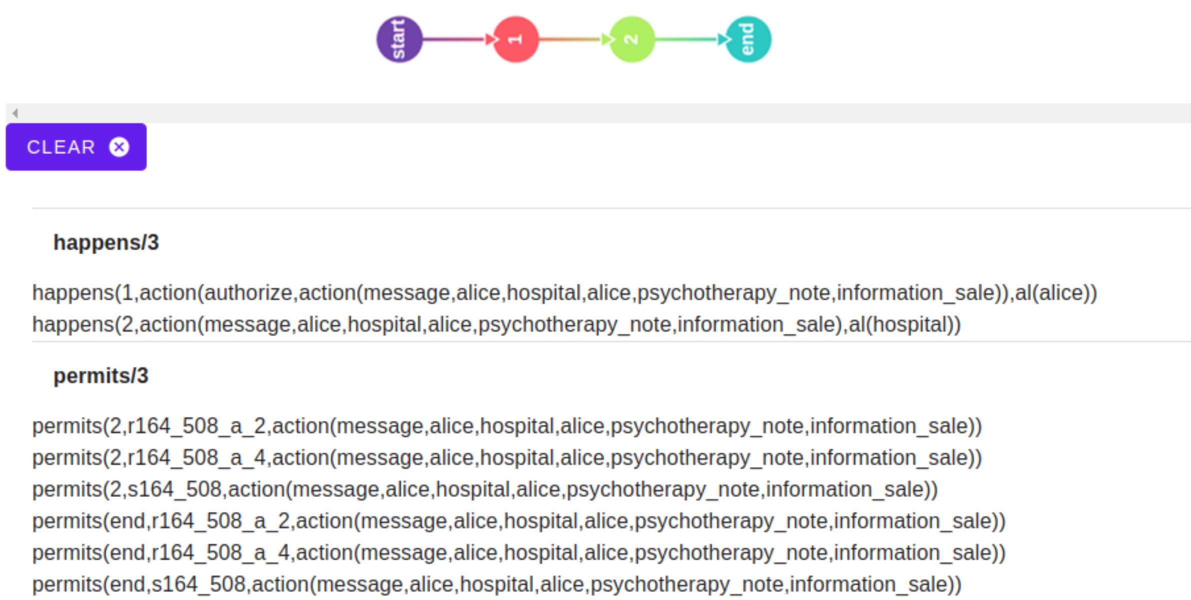

Figure 4.4: Screenshot of the output for Example 1 (most predicates hidden)

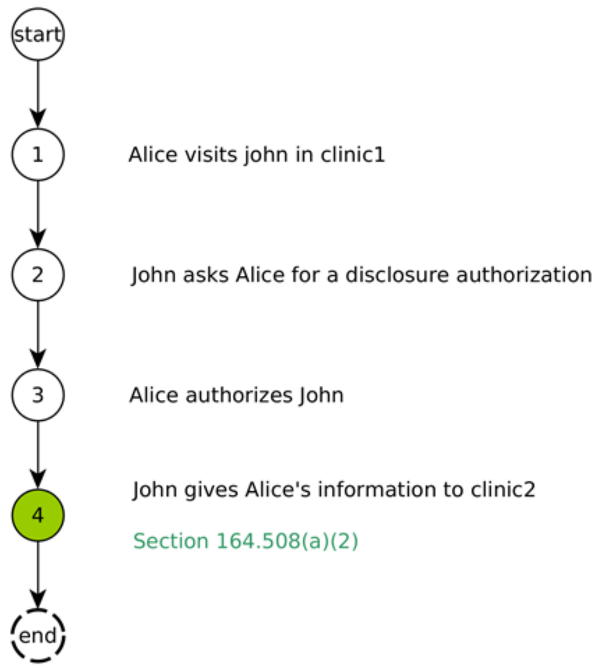

Figure 4.5: Example 1, with latent actions

Figure 4.4 shows a screenshot of our output visualization tool for the above encoding. The graph represents the timeline of generated events, in this case two actions. The `action` predicate takes the information type (psychotherapy notes) as well as the purpose of use (information sale). The `happens` predicates show which actions were taken, and the `permits` predicates show which clauses of HIPAA permitted those actions. We use output templates to translate the `happens` predicates into a human readable format:

1. alice authorizes hospital to sell their PHI to advertiser

2. hospital sells alice's PHI to advertiser

Note that the `permits` section shows six different predicates. This is because the action is permitted by both clause §164.508(a)(2) as well as clause §164.508(a)(4). The third predicate denotes that the action is permitted in aggregate by section §164.508. There are two sets of these three predicates, one for each time step that the action is permitted in. The action is only allowed time step 2 onwards, after the authorization has been made. Also note that for this query, the latent actions of "visiting the clinic" and "asking for authorization" were not added to the domain. The required fluents were instead part of the initial state in the query. Figure 4.5 shows another version with the latent actions.

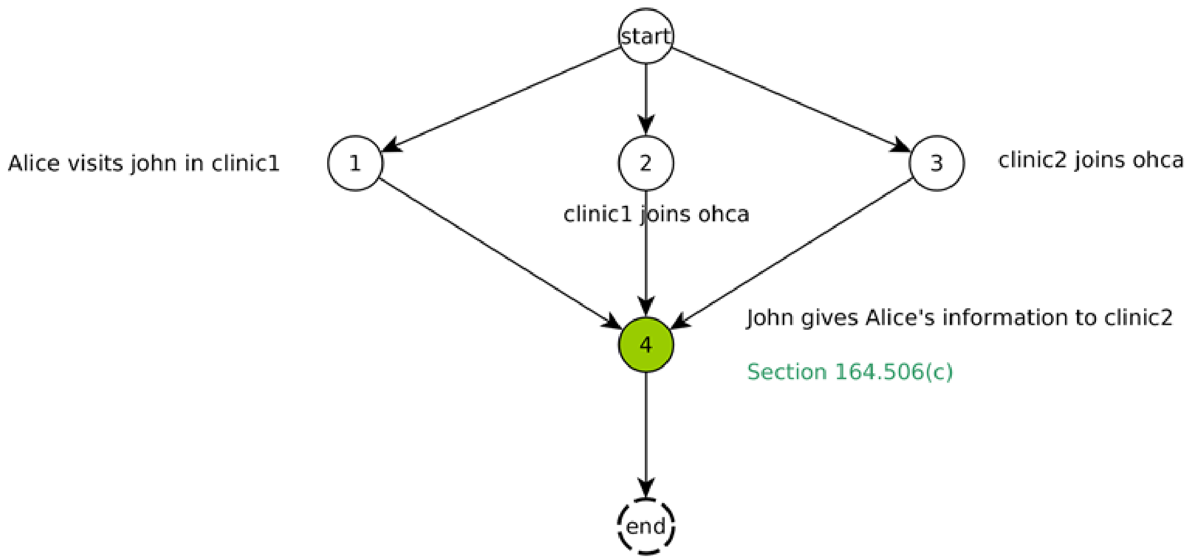

Figure 4.6: Example 2

4.5.1.2 Example 2: Follow-up question

The generated narrative in the first example shows the user explicitly agreeing to their data being sold. That is just one of the many possibilities that the planner generates. A natural follow-up question then might be whether it's possible for the data to be sold without the user's authorization. We can find this in the generated narratives manually, or we can add this constraint by simply adding the following line to Example 1 to narrow down the output:

Listing 4.6: Example 2.1: No patient authorization

```
% constraints
:-happens(_,action(authorize,_,al(alice))
```

This forbids any authorization by Alice.

Figure 4.6 shows one of the satisfying plans. The clinic visited by Alice joins an *Organized Healthcare Arrangement* which another clinic is also a part of. This allows the first clinic to transfer Alice's information to them without her authorization:

> 164.506(c) **Implementation specifications: Treatment, payment, or health care operations**
>
> 5. A covered entity that participates in an organized health care arrangement may disclose protected health information about an individual to other partici-

pants in the organized health care arrangement for any health care operations activities of the organized health care arrangement.

It is clear from the partial order graph that Alice's visit to her clinic could have *preceded* the clinic joining the Organized Healthcare Arrangement, which may be important information when making a privacy-conscious decision.

We can ask another follow-up question, disallowing involvement of other covered entities by adding `noautorole.` to the query to disable entities with roles being automatically generated. This results in an `UNSATISFIABLE` output, meaning no such plans are possible.

4.5.1.3 Example 3: Is an action allowed?

Alice's psychiatrist, John, is visited by a police officer, Bob. The officer demands Alice's records for an investigation. John wants to know if he can legally hand over Alice's PHI.

Listing 4.7: Example 3: Is an action allowed?

```
% constraints
role(bob, law_enforcement).
role(alice, person).
role(john, psychiatrist).
:- not happens(1, action(ask_info,alice,john,bob)).
:- not happens(2, action(message,bob,john,alice,_,_)).
before(1,2).
```

The planner generates a sequence where Bob shows John a court order demanding Alice's PHI before John hands over the information. This is allowed under HIPAA section 164.512(f)(1).

4.5.1.4 Example 4: Actions needed for compliance

Alice's psychiatrist wants to user her information for training interns. He wants to know if an authorization is required for this purpose. The planner generates Figure 4.8, where no authorization action appears. This is because HIPAA section 164.506(c)

4.5.1.5 Example 5: Intentional agents

The user wants to know under what circumstances their employer might obtain their personal health information. We give the user (Bob) and the employer (Jane) two antagonistic

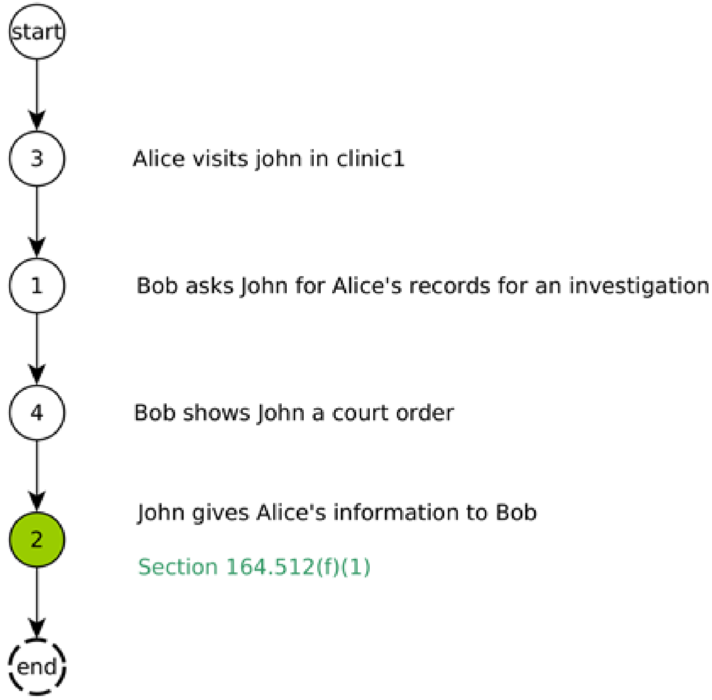

Figure 4.7: Example 3

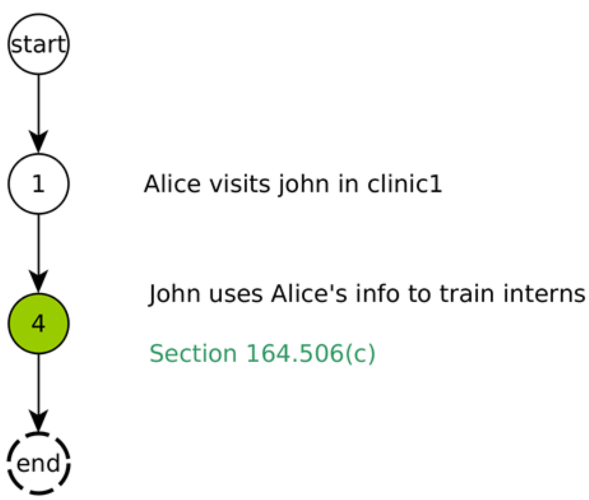

Figure 4.8: Example 4

intentions: the user wants to protect their info, while their employer wants to acquire the info.

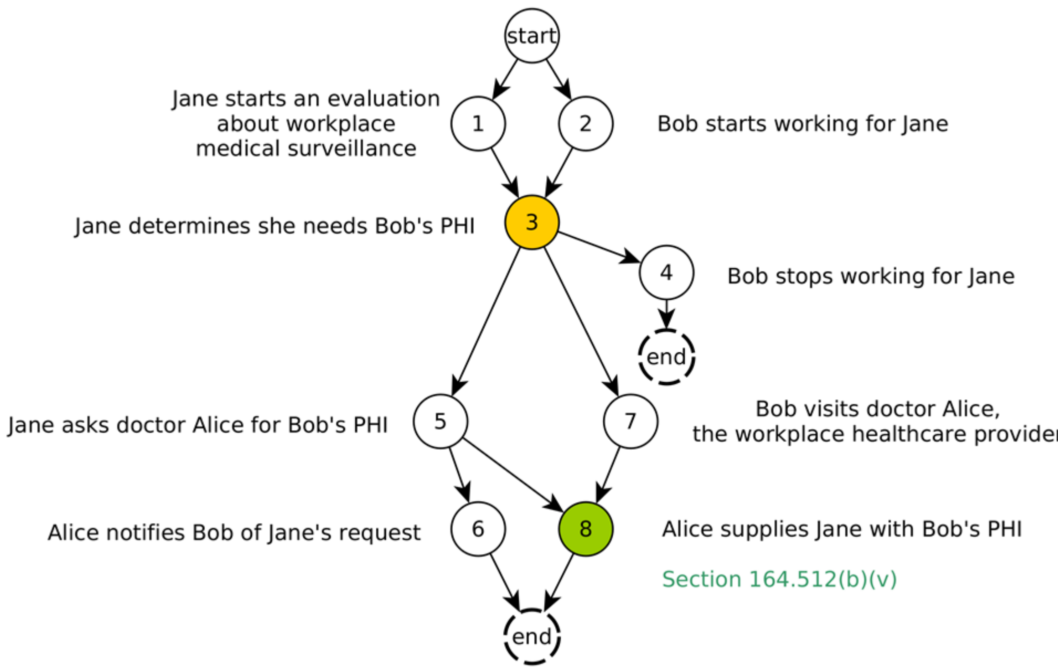

Figure 4.9: Example 5

One possible plan is shown in Figure 4.9. The planner says that the action of disclosing the information is permitted by §164.512(b)(v)(A), as part of workplace medical surveillance. However, if Bob quits his job before the info is requested, the doctor cannot disclose the info to the employer. The conflict between intentions occurs at node 3, where our "timelines" diverge. In one scenario, Bob visits Alice, and his information from that visit is disclosed to Jane. In another, Bob quits before this can happen.

4.5.2 Policy design

Here we demonstrate some ways in which our system can help policy design. Instead of providing full output graphs for each scenario, we will just provide information on how to encode each and a description of the output.

4.5.2.1 Example 6: Testing policy changes

The policy designer wants to test what effect a change in the policy would have. To encode this scenario, we encode both versions of the policy as described before, and give them different names (R1 and R2). Then, we add the following constraint:

Listing 4.8: Example 6

```
compliant_old(I, A) :-
    not forbids(I, R1, A): rule(R1, _),
        R1 != new_clause;
    permits(I, R2, A), rule(R2, _),
    R2 != new_clause.
compliant_new(I, A) :-
    not forbids(I, R1, A): rule(R1, _),
        R1 != old_clause;
    permits(I, R2, A), rule(R2, _),
    R2 != old_clause.
:- not happens(I, A, _),
    compliant_new(I, A),
    not compliant_old(I, A) : id(I).
```

As an example, we consider §164.508(a)(2), which has a `requirement` of obtaining an authorization before disclosing psychotherapy notes. If we change the `requirement` to a `category`, we find that this activates a different rule §164.506(c)(1), enabling disclosure of information, and our planner generates a narrative to that effect.

4.5.2.2 Example 7: Policy unit tests

As mentioned before, our system can also be used to verify narratives instead of generating them. It does so by attempting to generate a matching narrative.

Say we have a sequence of actions, A_1, A_2, ..., A_n. We want to check if these actions, if carried out in that order, violate any part of the law. For this use case, it is reasonable to assume that only a partial list of the actions may be available. The planner should be able to fill in the gaps to produce a complete narrative. We would encode this use case as in Listing 4.9.

Listing 4.9: Example 7

```
:- not happens(1, A_1_spec, _).
```

```
:- not happens(2, A_2_spec, _).
...
:- not happens(n, A_n_spec, _).

before(I,J) :- index(I), index(J), I<J.
:- not forbids(I, _, A), happens(I, A, _)
    : index(I).
```

The above technique forces the planner to generate actions A_1 through A_n at graph node indices $1...n$. It then says that the actions must be in order (though it allows other actions to be taken in between). Then it says that at least one of the actions must be forbidden by a rule.

This method could be used to write unit tests for a policy, encoding scenarios that must or must not be allowed. The tests could be run every time any changes are made to the policy.

4.6 Comparison with Other Presentation Methods

In this section, we will take the example of a fictional video on demand app called *Webflix*. We will see a few different representations of Webflix's privacy policy, and compare them to how an interactive system can benefit users.

> *Webflix* provides access to three kinds of programming: free, paid, and third-party. Users can watch free shows without having to create an account, and in that case, Webflix collects minimal amount of information. To watch paid shows, users can either purchase a subscription, or enable advertising to help support the platform. Webflix also provides access to shows from third-party publishers as a convenience. Users need to purchase additional subscriptions for these publishers, which can also be done from within the app. Some third-party publishers require Webflix to share certain types of user data with them. Additionally, Webflix uses search history and other kinds of user interactions with the app to personalize show recommendations. This can be disabled in app settings.

4.6.1 Apple's Privacy Labels

Apple introduced Privacy Labels in 2020. Developers are required to answer a survey describing what user data they collect and how they use it. This information is then presented to the user in an easy-to-understand, categorized manner, as part of an app's listing in the App Store. We have created a mock-up of what our fictional app's privacy policy might look like in this scheme, shown in Figure 4.10.

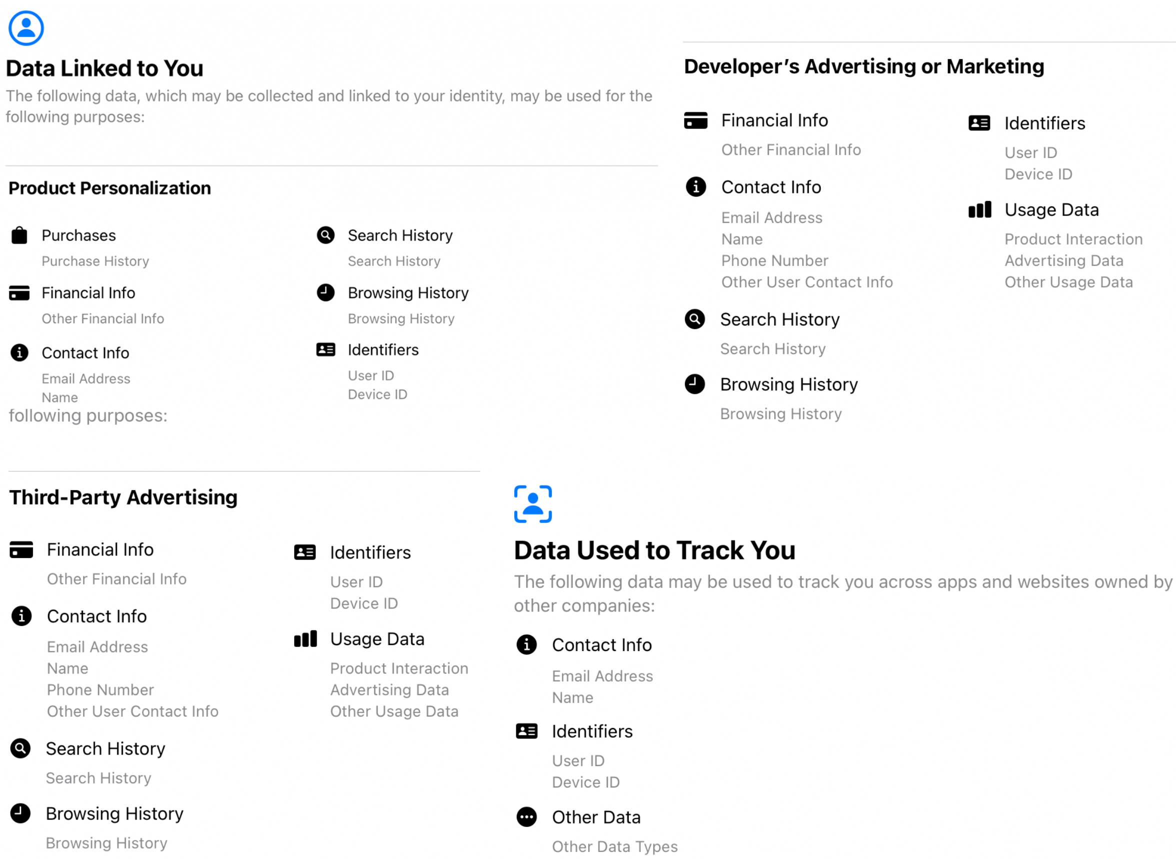

Figure 4.10: Apple's privacy labels for Webflix

This representation provides a summary view of the privacy policy. However, this over-represents the privacy risk for many of the app's users, since some of the data collection is dependent on users' actions within the app, like purchasing third-party subscriptions.

4.6.2 Privacy nutrition label

Privacy nutrition labels provide information about how the collected information is used, divided into a few basic categories. They also state whether the collection is opt-in or opt-out.

A mock-up of privacy nutrition label for our example is given in Figure 4.11. The label is an improvement over Apple's privacy labels in that it specifies where a user action could opt them out of information collection of a certain kind. However, it does not specify the kind of action needed to be taken. Being static, it also doesn't take into account how the collection can change based on user activities, like creating an account or making a purchase.

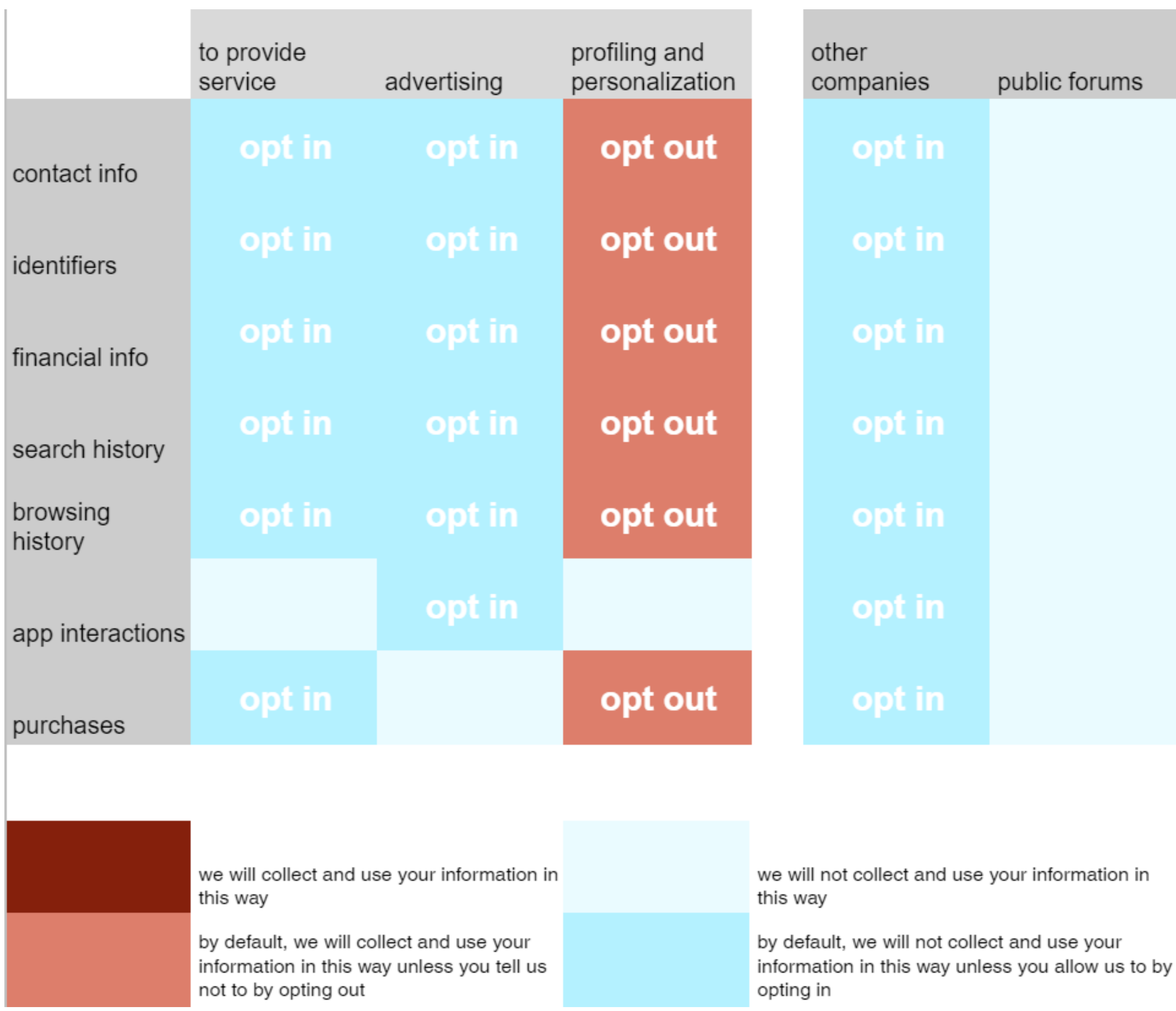

Figure 4.11: Privacy Nutrition Label (CMU)

4.6.3 Our System

Our system makes it possible to present the privacy policy in a more interactive way, allowing users to ask about their particular situation and also ask follow-up questions. Here we give a few examples of the kinds of questions a user might ask that cannot be answered with static presentation methods. We present some of the generated solution graphs in textual format for brevity.

Question "Under what circumstances will Webflix share my data with 3rd parties?".

Answer Our planner generates two different possibilities:

- The user watches premium content without having purchased a subscription
- The user purchases a subscription to a 3rd party publisher through the app

Question A follow-up can be "How do I watch premium content without my data being shared with 3rd parties?".

Answer The planner generates only one possibility:

- The user watches premium content after having purchased a subscription

Question "How do I prevent Webflix from collecting any data, while watching premium content?"

Answer

- The user purchases a subscription, turns off personalization in app settings, and then watches premium content

CHAPTER

5

LLM-ASSISTED DOMAIN ENCODING METHODOLOGY

RQ 3: *How do we automate extraction of a consistent planning domain from a natural language policy?*

5.1 Motivation

In the previous chapter, we showed that privacy policies inherently encode planning-like structures, where data flows through different states and processes according to specific conditions, rules, and temporal sequences. One major hurdle in implementing that approach is the tedious process of manual domain encoding.

We posit that Large Language Models (LLMs) are uniquely suited to this task. Firstly, they have been shown to have the capacity for understanding complex narrative structures, temporal reasoning, and multi-entity relationships. Secondly, LLMs have remarkable in-context (few-shot) learning abilities, allowing them to learn from and follow given natural language examples (Brown et al. 2020). Thirdly, privacy policies often make the implicit assumption that the reader has common knowledge about the world. For example, the understanding that a patient must first see a doctor before a doctor-patient relationship can be established. LLMs are trained on diverse textual sources and demonstrate this understanding.

In this chapter, we present a methodology to automate a large part of the domain encoding process.

5.2 Related Work

Automated analysis of privacy policies and generation of formal planning domains from natural language specifications are both active areas of research. Although no existing work directly addresses the translation of privacy policies into narrative planning domains, several related efforts provide important context for our approach.

5.2.1 Knowledge Graph and Semantic Representations

Bui et al. Bui et al. (2021) developed neural sequence-labeling models that extract fine-grained personal data phrases and their corresponding data practices.

PoliGraph (Cui et al. 2023) constructs knowledge graphs that capture statements in privacy policies as relations between different parts of the text. The subsequent PoliGrapher-LM extends this approach using large language models, demonstrating improved coverage in extracting complex relationships. Although this system creates structured representations and can detect contradictions within policies, it focuses on static analysis rather than enabling dynamic reasoning about privacy scenarios. It also does not capture all the details required to generate a full narrative domain.

5.2.2 Formal Logic and Ontology-Based Approaches

Torre et al. (2020) describe an approach for AI-assisted compliance checking of privacy policies against GDPR. This involves a conceptual model of the policy and GDPR, but doesn't contain all the information necessary to create a planning domain.

The W3C Data Privacy Vocabulary (DPV) (J. Pandit et al. 2024) provides a standardized ontology for representing privacy and data protection concepts. Complementary ontologies like PrOnto extend this foundation with specialized vocabularies for legal reasoning and consent management under GDPR. For our purposes, these ontologies are usually overkill, and lead to planning space explosion.

5.2.3 Natural Language to PDDL Translation

PolicyLR (Hooda et al. 2024) maps a pre-determined set of atomic statements to truth values according to the policy text. This can answer many privacy-related questions, but doesn't allow extraction of a fully executable narrative domain.

Liu et al. (2023) provide a way to extract planning problems, including domains and queries, from natural language text. However, the system seems to rely on highly structured natural language definitions of the domain, and did not perform well on privacy policies.

5.3 System Overview

We break down the task of domain encoding into several subtasks. These subtasks are somewhat in line with the manual encoding method mentioned in Chapter 4, but modified to fit the capabilities of LLM agents. We propose an agentic system with procedural logic steps to check for issues and domain convergence.

5.3.1 Global Context Extractor

Due to token limits (see 5.4.3), we must encode the domain in multiple pieces. The global context extractor ensures that all subsequent pieces use consistent terminology. It collects all mentioned entities, gives them canonical names and categories (Figure 5.2). It also detects synonyms of mentioned concepts, listing them and giving them unique names. See B.1.2 for the exact schema.

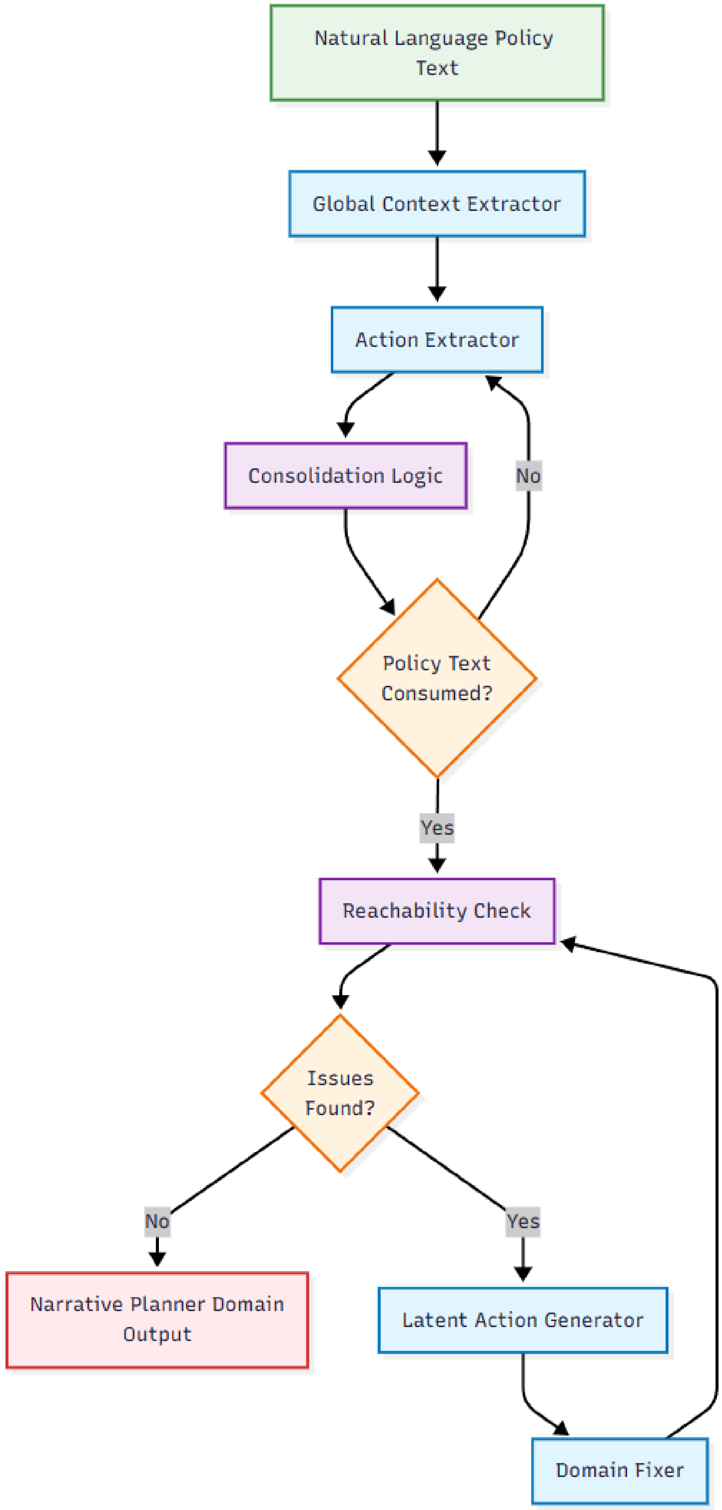

Figure 5.1: System block diagram

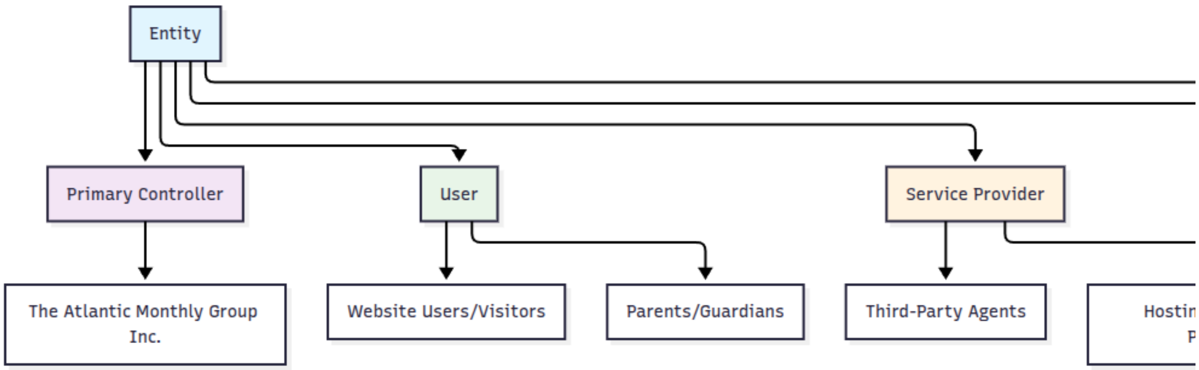

Figure 5.2: Part of entity hierarchy for a privacy policy from OPP-115

5.3.2 Action Extractor

This task extracts the requested number of actions from those mentioned in the policy, along with the following details:

1. Unique action name
2. Actor
3. Preconditions
4. Effects
5. Fluent parameters for preconditions and effects
6. Justification (spans from original policy where this action was derived from)

```
{
  "name": "administer_third_party_mailing",
  "actor": "atlantic",
  "preconditions": [
    "subscribed_to_publication(User)",
    "not opted_out_data_sharing(User)"
  ],
  "effects": [
    "administered_third_party_mailing(atlantic, User)"
  ]
},
```

Figure 5.3: Example extracted action

Actors are entities that have the choice to take the given action. For actions which don't have a clear actor, e.g. an earthquake, we define a special actor `environment`.

Fluent parameters must distinguish between planning variables vs. specific values (5.4.2.2). For example, in Figure 5.3, `User` starts with an upper case letter, while `atlantic` does not. This is because the policy can apply to multiple users, so it must be treated as a variable.

The justification text is important for debugging and as context for agents in later steps.

5.3.3 Consolidation Logic

The consolidation logic takes the existing set of context (entities, fluents, actions), and merges with it each successive segmented run of the Action Extractor.

5.3.4 Identifying Domain Errors

At this point, we have a rough outline of the domain, but there are usually several issues left to fix. In this part, our job is to identify common issues and list all the information needed to fix them.

5.3.4.1 Action reachability check

One very common form of domain error is when certain listed actions can never be executed, due to their preconditions being impossible to meet. To identify these issues, we use procedural code to create an action graph, where each node is an action and each edge is an effect of an action that matches with a pre-condition of another.

We say an action is reachable, if at least one other action can satisfy each of its preconditions. The following must be taken into account:

1. If a pre-condition parameter is a specific value (see 5.3.2), then it must be satisfied by an effect that has the same specific value, or a variable that can take that value.

2. An effect variable can take values passed as preconditions, or from the actor field. For example, if the effect is `opt_out_marketing(User,atlantic)`, then the values for `User` either come from a pre-condition that also mentions the same variable, or from the `User` being the actor of the action.

3. If a pre-condition parameter is a variable, it can take any value.

We mark unreachable actions along with the reason, as an input to the Domain Fixer.

5.3.4.2 Latent Action Generator

This agent proposes actions to produce fluents that satisfy preconditions for unreachable actions. An example could be `visit_doctor(Doctor,Patient)`, which activates the fluent denoting a doctor-patient relationship, as a precondition to executing other actions. These actions are required to form a full narrative graph, even if not mentioned directly in the policy text. In this case, a missing action is generated, with the appropriate parameters (actor, preconditions, etc.). To prevent infinite regress where new actions require new preconditions, which in turn require new actions, we limit the causal depth to 3, counting from actions that were generated by the Action Extractor.

5.3.5 Domain Fixer

Since actions are extracted in batches, the agent can come up with somewhat different names for the same concept, like `opt_in_marketing_emails(?user)` vs `subscribe_mailing_list (?user)`. The agent normalizes the symbols according to the definitions in the Global Context, if needed. It also de-duplicates actions based on schema key, or unions them based on the original text/justification.

5.4 Iterative Design Approach

5.4.1 Overall system evolution

Our overall system went through following phases of evolution based on feedback from experimentation:

1. Single prompt to do everything (poor performance)
2. Extract → Analyze → Fix (ran into token limits)
3. Iterative-extract → Analyze → Fix (not all issues fixed)
4. Iterative-extract → Analyze → Iterative-fix (current system)

Throughout this, we also analyzed the performance of individual components, tweaking agent prompts, schema, and their available context to fix issues with individual parts of the system (Figure 5.4).

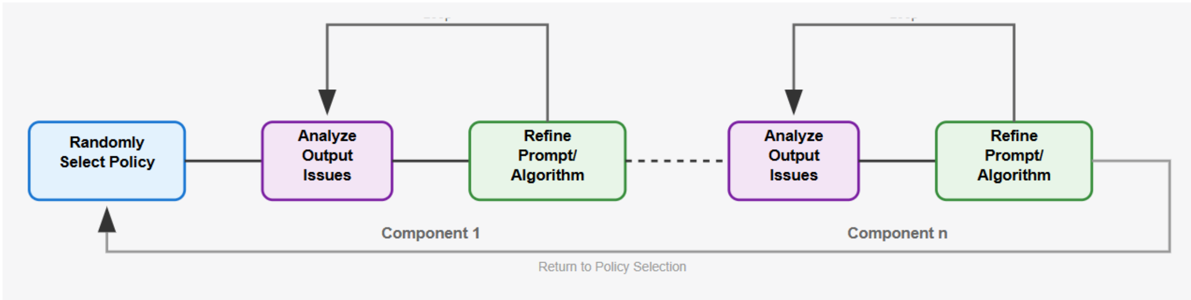

Figure 5.4: System design approach

Through this process, we maintained the following:

1. **Strict input/output schema:** This seemed to greatly increase output consistency. It also makes analysis of intermediate output significantly easier. A well-defined schema is also required by the procedural components.

2. **Just enough context:** Giving agents too much or too little information resulted in poorer output. If agents need to sift through a large context, they tend to miss vital information.

3. **Procedural code** as a verification checkpoint to predictably enforce certain properties on the output. As part of a loop, the procedural code informs the agent of what changes need to be made to nudge it closer to the desired output.

5.4.2 Prompt Refinement

Let's go through refinements of the Action Extractor agent prompt, along with justifications for each change. For the full prompt, see Appendix B.1.

5.4.2.1 Closed world assumption

Initial iterations of the prompt were struggling to merge different deontic modalities (e.g. forbidden, with an exception). As stated before, our planner makes a strong closed world assumption. Unless an action is encoded and allowed, we assume it cannot occur. This also leads to major deontic simplification, since we do not need to track forbidden actions (everything is forbidden by default). This significantly increased the success rate. We instruct the agent to discard forbidden actions:

> Closed world assumption: If an action/operator is not present in the final domain, it is considered forbidden. If a fact is not in the state, it is false. Forbidden actions can be left out of the domain.

5.4.2.2 Variable/constant distinction

To correctly encode actions, a distinction must be made between specific constants (e.g. "email") and variables which can take constants of a specific type (e.g. ?data). To force the agents to always make this distinction, we explicitly added it to the output schema, and added the following to any agent that generates parameters:

> Variable/Constant distinction: In every fluent argument, either supply a variable (e.g., '?dt') and declare its type in 'parameters', or supply a constant (e.g., 'email'). Never leave a slot ambiguous. '?actor' is a special variable to refer to the actor of an action.

5.4.2.3 Resource and permission tracking

Another area where the agent was struggling was actions requiring specific role checks or resources. For example, to share data, the actor must first have access to the data. If the action requires the actor to have a specific role ("only lawyers can access the database"), then this should be checked too in a standardized way. We add two special fluents to track data possession and roles:

> Fluent: A predicate like 'has_data(holder, user, email)' or with variables like 'has_data(holder, user, ?dt)'. All notions of posessing information should be normalized to the fluent 'has_data(?data_holder, ?data_owner, ?data_type)'. All role checks should be normalized to 'is(?entity, ?role)', like 'is(?actor, lawyer)'.

A related issue arises when the wording suggests "granting" permission to an actor. In our planner, this is equivalent to the existence of an action with a role guard. So we explicitly instruct the agent to convert these "meta-actions" to simple "guarded actions":

> Guarded event: A real-world action/event that includes role/resource constraints as preconditions. For example "only a company lawyer is permitted to access user data" should become an 'access_data' action, with the preconditions that the company possesses said data, and the actor is a laywer; and the effect that the lawyer now has said data.

5.4.3 Dealing with token limits

We quickly realized during experimentation that we were running into output token limits when dealing with larger policies. We tried two different paths to combat this:

1. Iterative extraction from the entire context

2. Global context with segmented extraction

With (1), we tried giving the main agent (Action Extractor) the entire context from the previous runs, as well as the entire policy text, while asking it to extract a limited number ($\tilde{1}5$) of actions not previously extracted. We were relying on additional cues in the context (justifications/source spans) to have the agent generate novel actions on each run.

With (2), we took a different approach, of generating a global context with disambiguation rules and global defaults. We then feed segments of the policy to the agent along with this global context, relying on it to capture long-range associations in text. This seemed to produce fewer duplicates, and did not lead to convergence issues, unlike the first approach.

Together, these changes led us to the design described in Figure 5.1.

5.5 Experimentation with smaller models

One of the downsides to the described methodology is that it can be cost-prohibitive on larger policy texts, due to repeated API calls until convergence. So, a natural question is whether smaller and cheaper models can perform just as well as frontier models on this task.

To answer this question, we attempted to run our system on Anthropic's Claude Haiku 3.5, as well as OpenAI's GPT-4o mini. Unfortunately, we experienced the same issues with both models:

1. Inability to follow output schema: Both models failed at various points to adhere to the given schema, polluting the output with irrelevant information. Figure 5.5 shows the model adding contact information to the entity hierarchy

2. Large number of missing actions: Even with smaller policies, the agents missed extracting >50% of essential actions, making the domain unusable.

3. Occasional hallucinations: Models occasionally created actions or preconditions that were not part of the given text, even when the call was well within token limits.

```json
  "entities": {
    "actor": {
      "company": {
        "name": "Mohegan Sun",
        "type": "Casino and online service provider",
        "primary_contact": {
          "phone": "1.888.226.7711",
          "email": "information@mohegansun.com",
          "address": "1 Mohegan Sun Boulevard Uncasville, CT 06382"
        }
      },
```

Figure 5.5: Claude Haiku 3.5 fails to adhere to output schema

Please refer to Appendix A.2 for a sample output from a small model. Compare it to Appendix A.1 from a larger model.

These results suggest that this task requires frontier model capabilities, since switching to small models didn't just degrade performance, but led to a complete task failure.

5.6 Evaluation

5.6.1 Existing Privacy QA Datasets

There are a few existing datasets potentially applicable to our system for testing.

Wilson et al. (2016) created the OPP-115 corpus, containing 115 privacy policies with fine-grained annotations across 10 data practice categories. This dataset has become a benchmark for privacy research.

Based on OPP-115, Ahmad et al. (2020) introduced one of the first comprehensive datasets for privacy policy question answering, PolicyQA, containing human-annotated question-answer pairs derived from real privacy policies. However, the answers do not directly address the question, but are excerpts from the original policy text, meant to contain the answer.

PrivacyQA (Alkhattabi et al. 2022) contains 1750 questions with expert annotations based on 35 mobile app privacy policies. However, the full text of the original policies isn't available. Only the excerpts that are deemed to contain the answers are available with the corresponding questions.

5.6.2 Approach

The lack of availability of full policy text rules out usage of PrivacyQA. Directly using policies from the original apps is not viable, since they could have been changed since the creation of the dataset.

Using OPP-115/PolicyQA poses its own problems: since the answers are in the form of annotations and excerpts, there is no easy way to automatically check whether a narrative graph matches with a given answer.

We will evaluate our system in two different ways:

1. **Domain encoding accuracy.** We manually encoded 5 randomly selected policies from the OPP-115 corpus as ground truth, to compare against the automatically encoded domain.

2. **End-to-end testing.** To test the entire system (domain extraction + planner), we randomly selected 10 policies from the PolicyQA corpus, along with 10 questions each, for a total of 100 queries. We then ran them against extracted domains (after any minimal syntax fixes required), manually comparing the planner output against the given answers.

```
{
  "name": "restrict_database_access",
  "actor": "company",
  "preconditions": ["information_stored_database(User)"],
  "effects": ["database_access_restricted_marketing_managers(User)"]
},
{
  "name": "access_database",
  "actor": "Person",
  "preconditions": [
    "information_stored_database(User)",
    "is(Person, marketing_manager)"
  ],
  "effects": ["has_data(Person, User, user_data)"]
}
```

Figure 5.6: Incorrectly extracted action (top) vs correct manually encoded action (bottom)

```json
{
  "name": "verify_age_for_promotion",
  "actor": "company",
  "parameters": [
    {
      "var": "?user",
      "type": "user"
    }
  ],
  "preconditions": [],
  "effects_add": [
    "is_21_or_older(?user)"
  ],
  "effects_del": [],
  "justifications": [
    "All recipients of any online promotions must be 21 years of age
    or older. Age is verified upon contact of the specified winner."
  ],
  "schema_key": "verify_age_for_promotion|company|user"
}
```

Figure 5.7: A "frivolous" action, which does not contribute to answering privacy questions

5.6.3 Results

5.6.3.1 Domain Encoding Accuracy

We compared extracted actions with manually encoded ones for 5 randomly selected OPP-115 policies. We put them into 4 categories:

1. Total extracted
2. Frivolous - unneeded actions that won't generally interfere with planning (figure 5.7).
3. Missing - actions completely missing from the extracted domain
4. Incorrect - non-frivolous actions that do not capture the intent of the policy and need fixing (figure 5.6).

The results are shown in Figure 5.8 (per-policy) and Table 5.1 (totals).

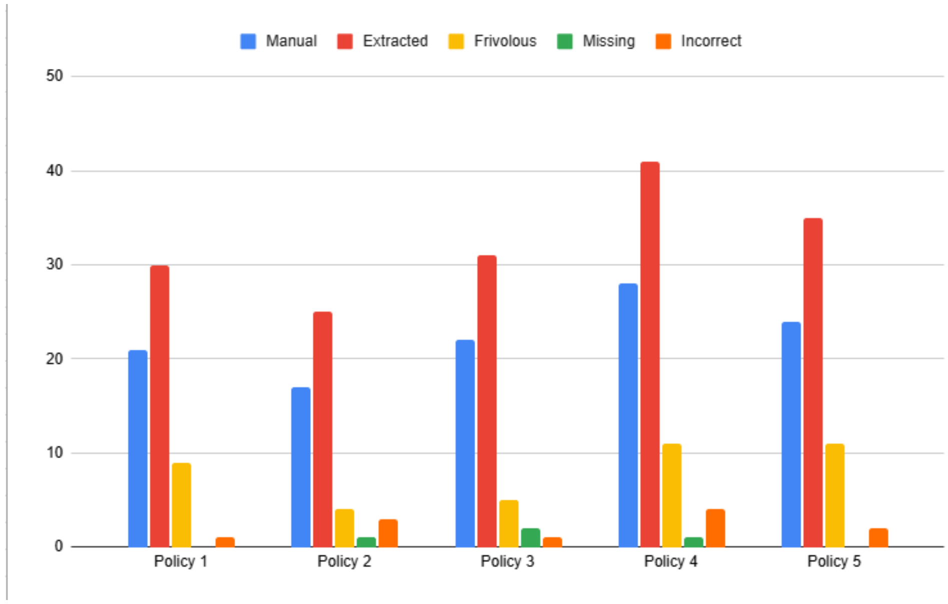

Figure 5.8: Actions for each encoded policy: manually encoded, extracted, frivolous, missing, incorrect

It should be noted that the "correctly" extracted actions need not be identical to the manually encoded ones. An action may be split into two causally linked sub-actions or vice versa, while preserving the intent of the policy. So, to do a fair assessment of the system's recall, we calculate:

$$\frac{manual - missing - incorrect}{manual} * 100$$

That gives us a recall of 86.6% according to table 5.1.

Table 5.1: Extracted vs Manually encoded

Category	**Total**	**Percentage**
Manual	112	100%
Extracted	162	144.6%
Frivolous	40	35.71%
Missing	4	3.57%
Incorrect	11	9.82%
Missing + Incorrect	15	13.4%

Large number of frivolous actions (relative to incorrect and missing) indicates that the system tends towards over-extraction rather than under-extraction. This suggests we have a conservative bias, favorable for completeness, but also responsible for output noise. Frivolous actions can cause state space explosion in our ASP planner, leading to significantly higher processing times. They also eat into the graph node budget (recall that our planner requires a pre-set upper limit on the number of generated nodes) and make interpreting the output difficult.

A possible way to combat this could be to add additional restrictions on the Action Extractor agent, limiting what kind of actions should be modeled. Lowest hanging fruit would be to only model actions that result in transmission of data, or that are the only producer for a fluent needed to activate such an action. This would probably be a tradeoff with final end-to-end accuracy (5.6.3.2).

However, there is a broader question of whether these "frivolous" actions add anything of value to the solution space. The purpose of this work is not just to replicate formal policy modeling efforts with a different set of tools, but to give the user an explorable model of the policy, which necessarily involves actors and real-world actions. In that context, there may be value in maximizing the action set and giving the user a richer vocabulary to form queries, even if parts of the query do not affect the final answer.

5.6.3.2 End-to-end testing

End-to-end testing with a set of 10 policies from PolicyQA, with 10 questions each, resulted in an accuracy of 82%.

This seems to confirm our hypothesis that the system is capturing the core ruleset of the policy.

There is minimal drop-off in performance (86% to 82%) going from domain-only evaluation to end-to-end evaluation. This suggests the *planning component* works quite effectively, if given reasonable input domains.

5.6.3.3 Domain variance analysis

To test the variance in answers, we ran the system three times on the same policies. The result is shown in table 5.2. Most of the variance comes from frivolous actions. Actual recall ranges from 82.1-86.6%.

Table 5.2: Domain variance

Category	**Total**	**Percentage**
Manual	112	100%
Extracted	158-171	141-152%
Frivolous	35-52	31.2-46.4%
Missing	4-7	3.6-6.3%
Incorrect	10-13	8.9-11.6%
Missing + Incorrect	15-20	13.4-17.9%

A kind of variance not directly captured in the table comes from the system sometimes splitting/combining actions. E.g.:

- travel_location → visit_booth → signup_membership
- travel_to_booth → signup_membership

In the second case, the 'visit_booth' action is *not* counted as missing or incorrect, since the intent of the original policy is preserved. Usually, these start out as an extracted action (signup_membership) with a dangling precondition, which is fixed by the later part of the pipeline.

Looking at the table, we can say that Missing (3.6-6.3%) and Incorrect (8.9-11.6%) actions remain tight, even if the number of frivolous actions varies widely with each run. This seems to suggest that *core domain extraction* is relatively stable across runs, even if the system struggles to reign in peripheral actions.

The fact that Missing actions remain consistently lower than Incorrect actions seems to suggest that identifying relevant actions is easier than correctly modeling them. This may also just stem from our over-extraction bias.

5.7 Conclusion

This chapter presented a methodology for LLM-assisted extraction of planning domains from privacy policies. Our multi-agent system addresses the challenge of domain encoding by breaking down the task into manageable subtasks: global context extraction, action extraction, consolidation, error identification, and domain fixing.

The evaluation demonstrates that our approach achieves strong performance with 86.6% recall in domain encoding accuracy and 82% accuracy in end-to-end testing. The system shows a bias toward over-extraction, generating frivolous actions that don't interfere with core functionality but may impact processing efficiency. The minimal performance drop between domain-only and end-to-end evaluation (86% to 82%) confirms that the planning component works effectively.

An iterative design approach, emphasizing strict input/output schemas, appropriate context management, and procedural verification checkpoints, proved essential for achieving consistent results.

The methodology represents a step toward automating privacy policy analysis while maintaining the rich, explorable narrative structure that enables dynamic reasoning about complex privacy scenarios. Future work could focus on reducing frivolous action generation and exploring the trade-offs between action set completeness and computational efficiency.

CHAPTER

6

CONCLUSION

In this dissertation, we have investigated a novel approach to making privacy policies more comprehensible and explorable through narrative generation. By treating policies as logical rulesets that constrain narrative solution spaces, we have demonstrated that planning-based narrative generation serves as a viable alternative to traditional textual question-answering systems while enabling richer interaction with complex policy domains. We developed an incidents classifier in support of a privacy incidents database, capturing privacy violations beyond traditional security breaches.

In Chapter 3, we designed a narrative planner using Answer Set Programming, enabling encoding of arbitrary rulesets as constraints on the narrative space.

In Chapter 4, we developed a systematic methodology for encoding privacy regulations, demonstrated through a case study of HIPAA data transmission rules, showing how formal policy modeling can be made accessible through narrative contexts. The system allows multiple modes of interaction with the solution space, like asking "what if" questions, checking compliance of a given sequence of actions, and exploring consequences of changes to policies.

In Chapter 5, we created an LLM-assisted pipeline, alleviating the manual effort needed in encoding of privacy policies as planner constraints. The system achieved 86.6% recall of encoded actions, bringing our approach closer to real-world application.

Our work demonstrates that narrative generation provides a natural bridge between logical precision and human policy comprehension. By representing policies as constraints on narrative possibility spaces, we enable dynamic exploration of policy implications while preserving contextual information. The practical implications extend across stakeholder groups: users can explore policies through relevant scenarios rather than dense legalese; organizations can test policy changes and hunt for edge cases; regulators can assess policy impact across diverse scenarios.

REFERENCES

World's biggest data breaches, selected losses greater than 30,000 records (updated 11th august 2015). `http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks`.

A. Mamiit. Spotify revamps new privacy policy after user backlash. `http://developer.nytimes.com/`, September 5, 2015.

A. Southall. U.S. workers are on alert after breach of data. `http://www.nytimes.com/2010/11/07/us/07breach.html?_r=0`, November 6, 2010.

Lillian Ablon, Paul Heaton, Sasha Romanosky, and Diana Catherine Lavery. *Consumer Attitudes Toward Data Breach Notifications and Loss of Personal Information*. Rand Corporation, 2016.

Alessandro Acquisti and Jens Grossklags. What can behavioral economics teach us about privacy. *Digital privacy: theory, technologies and practices*, 18:363–377, 2007.

Alessandro Acquisti, Allan Friedman, and Rahul Telang. Is there a cost to privacy breaches? An event study. *ICIS 2006 Proceedings*, page 94, 2006.

Anne Adams and Martina Angela Sasse. Privacy in multimedia communications: Protecting users, not just data. In *People and computers XV—Interaction without frontiers*, pages 49–64. Springer, 2001.

Advisen. Cyber loss: Outsourcing data to the cloud doesn't diminish risk. `http://www.advisenltd.com/2015/09/10/cyber-loss-outsourcing-data-to-the-cloud-doesnt-diminish-risk`, September, 2015.

Wasi Uddin Ahmad, Jianfeng Chi, Yuan Tian, and Kai-Wei Chang. Policyqa: A reading comprehension dataset for privacy policies. *arXiv preprint arXiv:2010.02557*, 2020.

Khalid Alkhattabi, Davita Bird, Kai Miller, and Chuan Yue. Question answering models for privacy policies of mobile apps: Are we there yet? In *International Conference on Science of Cyber Security*, pages 333–352. Springer, 2022.

Grigoris Antoniou, David Billington, Guido Governatori, and Michael J Maher. On the modeling and analysis of regulations. 1999.

Article 29 Data Protection Working Party. Google privacy policy: WP29 proposes a compliance package. `http://www.cnil.fr/english/news-and-events/news/article/google-privacy-policy-wp29-proposes-a-compliance-package/`, September 2014.

Adam Barth, Anupam Datta, John C Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15–pp. IEEE, 2006.

Adam Barth, John Mitchell, Anupam Datta, and Sharada Sundaram. Privacy and utility in business processes. In *20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 279–294. IEEE, 2007.

Trevor JM Bench-Capon, Gwen O Robinson, Tom W Routen, and Marek J Sergot. Logic programming for large scale applications in law: A formalisation of supplementary benefit legislation. In *Proceedings of the 1st international conference on Artificial intelligence and law*, pages 190–198, 1987.

Anne-Gwenn Bosser, Pierre Courtieu, Julien Forest, and Marc Cavazza. Structural analysis of narratives with the coq proof assistant. In *International Conference on Interactive Theorem Proving*, pages 55–70. Springer, 2011.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001. ISSN 0885-6125. URL `http://dx.doi.org/10.1023/A:1010933404324`.

Selmer Bringsjord and David Ferrucci. *Artificial intelligence and literary creativity: Inside the mind of BRUTUS, a storytelling machine*. Psychology Press, 1999.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Jerome Bruner. The narrative construction of reality. *Critical inquiry*, 18(1):1–21, 1991.

Duc Bui, Kang G Shin, Jong-Min Choi, and Junbum Shin. Automated extraction and presentation of data practices in privacy policies. *Proceedings on Privacy Enhancing Technologies*, 2021.

William E. Byrd. *Relational Programming in miniKanren: Techniques, Applications, and Implementations*. PhD thesis, Indiana University, 2009.

Ryan Calo. The boundaries of privacy harm. *Indiana Law Journal*, 86(3):1131–1162, 2011.

Sherol Chen, Adam M Smith, Arnav Jhala, Noah Wardrip-Fruin, and Michael Mateas. Role-model: towards a formal model of dramatic roles for story generation. In *Proceedings of the Intelligent Narrative Technologies III Workshop*, page 8. ACM, 2010.

Omar Chowdhury, Andreas Gampe, Jianwei Niu, Jeffery von Ronne, Jared Bennatt, Anupam Datta, Limin Jia, and William H Winsborough. 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*, pages 3–14, 2013.

G. Cluley. VX Heavens, old-school virus-writing website, raided by police, March 28, 2012.

Lorrie Faith Cranor. P3p: Making privacy policies more useful. *IEEE Security & Privacy*, 1 (6):50–55, 2003.

Lorrie Faith Cranor. Necessary but not sufficient: Standardized mechanisms for privacy notice and choice. *J. on Telecomm. & High Tech. L.*, 10:273, 2012.

Lorrie Faith Cranor, Praveen Guduru, and Manjula Arjula. User interfaces for privacy agents. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 13(2):135–178, 2006.

Hao Cui, Rahmadi Trimananda, Athina Markopoulou, and Scott Jordan. {PoliGraph}: Automated privacy policy analysis using knowledge graphs. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1037–1054, 2023.

D. Palmer. ICO investigating 56 Dean Street clinic for disclosing details of 780 HIV patients in data breach. `http://www.computing.co.uk/ctg/news/2424278/ico-investigating-health-clinic-for-disclosing-details-of-780-hiv-patients-in-mass-email`, September 2, 2015.

Henry DeYoung, Deepak Garg, Limin Jia, Dilsun Kaynar, and Anupam Datta. Experiences in the logical specification of the hipaa and glba privacy laws. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 73–82, 2010.

E. Bailey Jr. Google in the hot seat over WiFi privacy breach. `http://www.huffingtonpost.com/2010/06/08/google-privacy-slammed-ov_n_604084.html`, June 8, 2010.

E. Steel and J. Vascellaro. Facebook, myspace confront privacy loophole. the wall street journal. may 21, 2010. `http://www.wsj.com/news/articles/SB10001424052748704513104575256701215465596`.

Markus Eger and Chris Martens. Character beliefs in story generation. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

L. England. People are freaking out because WHSmith is accidentally emailing customer contact details to other customers, September 2, 2015.

Enigma Software Group. Enigma software's threat database. `http://www.enigmasoftware.com/threat-database/`.

FTC. Agreement containing consent order in the matter of facebook, inc, a corporation. `https://www.ftc.gov/sites/default/files/documents/cases/2011/11/111129/facebookagree.pdf`. File no. 092 3184. United Stats Of America Federal Trade Commission.

FTC. Eli Lilly settles FTC charges concerning security breach. `https://www.ftc.gov/news-events/press-releases/2002/01/eli-lilly-settles-ftc-charges-concerning-security-breach`, January 18, 2002.

Simson Garfinkel and Mary Frances Theofanos. A collection of non-breach privacy events, February, 2016.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. A user's guide to gringo, clasp, clingo, and iclingo. 2008.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.

Richard J Gerrig and William G Wenzel. The role of inferences in narrative experiences. *Inferences during reading*, page 362, 2015.

Nathaniel S Good and Aaron Krekelberg. Usability and privacy: a study of kazaa p2p file-sharing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 137–144. ACM, 2003.

Guido Governatori, Vineet Padmanabhan, Antonino Rotolo, and Abdul Sattar. A defeasible logic for modelling policy-based intentions and motivational attitudes. *Logic Journal of IGPL*, 17(3):227–265, 2009.

Amit Goyal, Ellen Riloff, and Hal Daumé III. A computational model for plot units. *Computational Intelligence*, 29(3):466–488, 2013.

The Guardian. Guardian open platform. `http://open-platform.theguardian.com/`. Accessed: 2016-3-3.

H. Brignull, M. Miquel, J. Rosenberg, J. Offer. Dark patterns: Fighting user deception worldwide. `http://darkpatterns.org/`.

Ashish Hooda, Rishabh Khandelwal, Prasad Chalasani, Kassem Fawaz, and Somesh Jha. Policylr: A logic representation for privacy policies. *arXiv preprint arXiv:2408.14830*, 2024.

Numaan Huq. Follow the data: Dissecting data breaches and debunking myths. *TrendMicro Research Paper*, September, 2015.

ITRC. Identity theft resource center. `http://www.idtheftcenter.org/`.

Harshvardhan J. Pandit, Beatriz Esteves, Georg P. Krog, Paul Ryan, Delaram Golpayegani, and Julian Flake. Data privacy vocabulary (dpv)–version 2.0. In *International Semantic Web Conference*, pages 171–193. Springer, 2024.

Daniel Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

Lennart C.L. Kats and Eelco Visser. The spoofax language workbench: Rules for declarative specification of languages and ides. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '10, pages 444–463, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0203-6. doi: 10.1145/1869459.1869497. URL http://doi.acm.org/10.1145/1869459.1869497.

Patrick Gage Kelley, Joanna Bresee, Lorrie Faith Cranor, and Robert W Reeder. A" nutrition label" for privacy. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, pages 1–12, 2009.

Casey Klein, John Clements, Christos Dimoulas, Carl Eastlund, Matthias Felleisen, Matthew Flatt, Jay A McCarthy, Jon Rafkind, Sam Tobin-Hochstadt, and Robert Bruce Findler. Run your research: on the effectiveness of lightweight mechanization. *ACM SIGPLAN Notices*, 47(1):285–296, 2012.

Spyros Kokolakis. Privacy attitudes and privacy behaviour: A review of current research on the privacy paradox phenomenon. *Computers & security*, 64:122–134, 2017.

Bert-Jaap Koops, Bryce Clayton Newell, Tjerk Timan, Ivan Škorvánek, Tom Chokrevski, and Maša Galič. A typology of privacy. *University of Pennsylvania Journal of International Law, Forthcoming*, 2016.

Robert Kowalski and Marek Sergot. A logic-based calculus of events. In *Foundations of knowledge base management*, pages 23–55. Springer, 1989.

L. Rao. Sexual activity tracked by Fitbit shows up in Google search results. http://techcrunch.com/2011/07/03/sexual-activity-tracked-by-fitbit-shows-up-in-google-search-results/, July 3, 2011.

Peifung E Lam, John C Mitchell, and Sharada Sundaram. A formalization of hipaa for a medical messaging system. In *International Conference on Trust, Privacy and Security in Digital Business*, pages 73–85. Springer, 2009.

Wendy G Lehnert. Plot units and narrative summarization. *Cognitive science*, 5(4):293–331, 1981.

Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

Jialiu Lin, Shahriyar Amini, Jason I Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 501–510, 2012.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

T. Lutz. Johnny Manziel asks for privacy as he enters rehab. *The Guardian*, February 2, 2015.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, 2008. ISBN 0521865719, 9780521865715.

Chris Martens, Anne-Gwenn Bosser, Joao F Ferreira, and Marc Cavazza. Linear logic programming for narrative generation. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 427–432. Springer, 2013.

Chris Martens, Joao F Ferreira, Anne-Gwenn Bosser, and Marc Cavazza. Generative story worlds as linear logic programs. In *Seventh Intelligent Narrative Technologies Workshop*, 2014.

Microsoft. Microsoft's malware protection center. `https://www.microsoft.com/security/portal/mmpc/`.

Erik T Mueller. Understanding goal-based stories through model finding and planning. In *AAAI Fall Symposium: Intelligent Narrative Technologies*, pages 95–102, 2007.

Pradeep K Murukannaiah, Chinmaya Dabral, Karthik Sheshadri, Esha Sharma, and Jessica Staddon. Learning a privacy incidents database. In *Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp*, pages 35–44, 2017.

NCSU, UNCC, Clemson University, and Rochester Institute of Technology. The privacy incidents database research project. `https://research.csc.ncsu.edu/privacyincidents/index.php`.

Helen Nissenbaum. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004.

NYTimes. The New York Times API. `http://developer.nytimes.com/`. Accessed: May 2016.

Paul N Otto and Annie I Antón. Addressing legal requirements in requirements engineering. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 5–14. IEEE, 2007.

P. K. Murukannaiah, J. Staddon, H. Lipford and B. Knijnenburg. PrIncipedia: A privacy incidents encyclopedia. Working paper at the 2016 Privacy Law Scholars Conference.

Tan Pang-Ning, Michael Steinbach, Vipin Kumar, et al. *Introduction to data mining*. Addison-Wesley Longman, Boston, 2006.

Julie Porteous, Marc Cavazza, and Fred Charles. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2):1–21, 2010.

PRC. Privacy Rights Clearinghouse. `http://www.privacyrights.org/`.

Robert W Proctor, M Athar Ali, and Kim-Phuong L Vu. Examining usability of web privacy policies. *Intl. Journal of Human–Computer Interaction*, 24(3):307–328, 2008.

Vladimir Propp. *Morphology of the Folktale*, volume 1. University of Texas Press, 1928.

R. Singel. Netflix spilled your brokeback mountain secret, lawsuit claims. wired, december 2009. `http://www.wired.com/2009/12/netflix-privacy-lawsuit/`.

Ranks NL. Default English stopwords list. `http://www.ranks.nl/stopwords`. Accessed: 2016-12-19.

Joel R Reidenberg, Travis Breaux, Lorrie Faith Cranor, Brian French, Amanda Grannis, James T Graves, Fei Liu, Aleecia McDonald, Thomas B Norton, and Rohan Ramanath. Disagreeable privacy policies: Mismatches between meaning and users' understanding. *Berkeley Tech. LJ*, 30:39, 2015.

Mark O Riedl and Robert Michael Young. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, 39:217–268, 2010.

Elena Rishes, Stephanie M Lukin, David K Elson, and Marilyn A Walker. Generating different story tellings from semantic representations of narrative. In *International Conference on Interactive Digital Storytelling*, pages 192–204. Springer, 2013.

Sasha Romanosky. Examining the costs and causes of cyber incidents. In *Twelfth Annual Forum on Financial Information Systems and Cybersecurity: A Public Policy Perspective*, January, 2016.

Sasha Romanosky, Rahul Telang, and Alessandro Acquisti. Do data breach disclosure laws reduce identity theft? *Journal of Policy Analysis and Management*, 30(2):256–286, 2011.

Elizabeth Irvin Ross. *Write Now*. Barnes & Noble Publishing, 1993.

Grigore Roşu and Traian Florin Şerbănuţă. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010. doi: 10.1016/j.jlap.2010.03.012.

Salman Saghafi and Daniel J Dougherty. Razor: Provenance and exploration in model-finding. In *PAAR@ IJCAR*, pages 76–93. Citeseer, 2014.

Charlie Savage. F.b.i. violated rules in obtaining phone records, report says. `http://www.nytimes.com/2010/01/21/us/21fbi.html?_r=0`, January 20, 2010. Accessed: 2016-05-16.

Security Incidents Organization. RISI online incident database. http://www.risidata.com/Database.

K. Sheshadri, N. Ajmeri, and J. Staddon. No (privacy) news is good news: An analysis of privacy news in the u. s. and u. k. from 2010-2016. *Under review*, 2016.

Shey, Heidi. Market overview: Customer data breach notification and response services," forrester, august 5, 2015.

Alireza Shirvani, Rachelyn Farrell, and Stephen G Ware. Combining intentionality and belief: Revisiting believable character plans. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

Daniel J. Solove. A taxonomy of privacy. *University of Pennsylvania Law Review, Vol. 154, No. 3*, January 2006.

Symantec. Symantec security response threat writeups. https://www.symantec.com/security_response/landing/azlisting.jsp.

Richard Taupe, Antonius Weinzierl, and Gottfried Schenner. Introducing heuristics for lazy-grounding asp solving. In *1st International Workshop on Practical Aspects of Answer Set Programming*, 2017.

Emina Torlak and Rastislav Bodik. Growing solver-aided languages with rosette. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pages 135–152. ACM, 2013.

Damiano Torre, Sallam Abualhaija, Mehrdad Sabetzadeh, Lionel Briand, Katrien Baetens, Peter Goes, and Sylvie Forastier. An ai-assisted approach for checking the completeness of privacy policies against gdpr. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 136–146. IEEE, 2020.

U. S. Government Accountability Office. Data breaches are frequent, but evidence of resulting identity theft is limited; however, the full extent is unknown, 2007. Report to Congressional Requesters. Washington, D.C., GAO-07-737.

US-CERT. United States Computer Emergency Readiness Team. https://www.us-cert.gov/.

Andre Valente and Joost Breuker. On-line: An architecture for modelling legal information. In *Proceedings of the 5th international conference on Artificial intelligence and law*, pages 307–315, 1995.

Verizon. 2015 data breach investigations report, 2015.

Anthony J Viera and Joanne M Garrett. Understanding interobserver agreement: The kappa statistic. *Fam Med 37, 5 (2005)*, pages 360–363, 2005.

Theo Wadsley and Malcolm Ryan. A belief-desire-intention model for narrative generation. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

Stephen G Ware and R Michael Young. Cpocl: A narrative planner supporting conflict. In *Seventh artificial intelligence and interactive digital entertainment conference*, 2011.

H. Warwick. Photographers – don't pap our wild animals, they need some privacy too. *The Guardian*, November 10, 2015.

Shomir Wilson, Florian Schaub, Aswarth Abhilash Dara, Frederick Liu, Sushain Cherivirala, Pedro Giovanni Leon, Mads Schaarup Andersen, Sebastian Zimmeck, Kanthashree Mysore Sathyendra, N Cameron Russell, et al. The creation and analysis of a website privacy policy corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1330–1340, 2016.

R Michael Young. Notes on the use of plan structures in the creation of interactive plot. In *AAAI fall symposium on narrative intelligence*, pages 164–167, 1999.

APPENDICES

APPENDIX

A

DOMAIN ENCODING OUTPUTS

A.1 Agent output samples

A.1.1 Global context extractor

Listing A.1: Global context agent (Sonnet 4)


```
{
  "definitions": {
    "personal_information": [
      "personal information",
      "information",
      "data",
      "personal data"
    ],
    "third_party": [
      "third party",
      "anyone"
```

```
    ],
    "email_updates": [
      "email broadcasts",
      "email correspondence",
      "online promotions"
    ],
    "database": [
      "database"
    ],
    "momentum_member": [
      "Momentum member",
      "Player's Club"
    ],
    "children": [
      "children"
    ],
    "unsubscribe": [
      "unsubscribe",
      "removed"
    ]
  },
  "global_defaults": [
    {
      "rule": "Company does not share or sell personal information to anyone",
      "source_spans": [
        "Mohegan Sun does not share or sell your personal information to anyone"
      ]
    },
    {
      "rule": "Company does not participate in SPAM or send unsolicited emails",
      "source_spans": [
```

```
        "Mohegan Sun does not participate in any format of SPAM and will not send any unsolicited emails"
      ]
    },
    {
      "rule": "Information is not shared with any third party",
      "source_spans": [
        "None of the information collected is sold or shared with any third party"
      ]
    },
    {
      "rule": "Site is not intended for children under 21",
      "source_spans": [
        "This site is not intended for children",
        "All recipients of any online promotions must be 21 years of age or older"
      ]
    }
  ],
  "entities": {
    "actors": {
      "company": "Mohegan Sun - casino and entertainment company operating mohegansun.com website",
      "user": "Website visitors and patrons who provide personal information online or at physical location",
      "marketing_managers": "Specific Marketing Managers at Mohegan Sun who have access to collected information",
      "customer_service": "Customer service department that handles information updates and corrections
```

```
      ",
    "parents_guardians": "Parents or legal guardians supervising children's use of the site",
    "children": "Individuals under 21 years of age who are restricted from using services"
  },
  "systems": {
    "website": "Mohegan Sun website located at mohegansun.com",
    "database": "Storage system for collected personal information, available only to Mohegan Sun",
    "players_club_booth": "Physical location at Mohegan Sun where patrons can sign up for Momentum membership",
    "email_system": "System used for sending email broadcasts and correspondence"
  },
  "data_types": {
    "name": "Personal name of individuals",
    "email_address": "Email addresses collected online and at physical location",
    "home_address": "Physical addresses of users",
    "phone_number": "Phone numbers collected at Player's Club booth",
    "areas_of_interest": "Data regarding user preferences and interests",
    "age_verification": "Information used to verify users are 21 or older"
  },
  "purposes": {
    "email_broadcasts": "Sending tailored information via email upon user request",
    "momentum_membership": "Player's Club membership program at Mohegan Sun",
```

```
      "marketing": "Marketing activities conducted by Marketing Managers",
      "customer_service": "Providing support and allowing information updates",
      "age_verification": "Ensuring compliance with 21+ age requirement for promotions"
    },
    "rights_mechanisms": {
      "unsubscribe_link": "Link included in every email allowing users to opt out",
      "information_update": "Ability to update or correct address/email by calling customer service",
      "customer_service_contact": "Phone number 1.888.777.7920 for information management"
    }
  }
}
```

A.1.2 Action extraction loop

Listing A.2: Action extraction loop (Sonnet 4)

```
{
  "entities_delta": {
    "actors": {
      "marketing_manager": "Marketing Manager at Mohegan Sun with access to collected information",
      "customer_service_rep": "Customer service representative who can update user information"
    },
    "data_types": {
      "areas_of_interest": "Data collected regarding user preferences and interests"
    },
```

```
    "systems": {
      "email_correspondence": "Email system used for sending broadcasts and communications"
    }
  },
  "fluents_delta": [
    "has_data(?data_holder, ?data_owner, ?data_type)",
    "is(?entity, ?role)",
    "requested_emails(?user)",
    "is_momentum_member(?user)",
    "is_21_or_older(?user)",
    "subscribed_to_emails(?user)",
    "at_location(?user, ?location)"
  ],
  "actions": [
    {
      "name": "collect_online_information",
      "actor": "company",
      "parameters": [
        {
          "var": "?user",
          "type": "user"
        },
        {
          "var": "?data_type",
          "type": "data_types"
        }
      ],
      "preconditions": [],
      "effects_add": [
        "has_data(company, ?user, ?data_type)",
        "has_data(database, ?user, ?data_type)"
      ],
      "effects_del": [],
      "justifications": [
```

```
        "The information collected online is used for email broadcasts utilizing tailored information upon the request of a visitor. Information received such as name, email address, home address and data collected regarding areas of interest is stored in a database."
      ],
      "schema_key": "collect_online_information|company|user,data_types"
    },
    {
      "name": "send_email_broadcast",
      "actor": "company",
      "parameters": [
        {
          "var": "?user",
          "type": "user"
        }
      ],
      "preconditions": [
        "has_data(company, ?user, email_address)",
        "requested_emails(?user)",
        "subscribed_to_emails(?user)"
      ],
      "effects_add": [],
      "effects_del": [],
      "justifications": [
        "The information collected online is used for email broadcasts utilizing tailored information upon the request of a visitor."
      ],
      "schema_key": "send_email_broadcast|company|user"
    },
    {
```

```
    "name": "access_user_data",
    "actor": "marketing_manager",
    "parameters": [
      {
        "var": "?user",
        "type": "user"
      },
      {
        "var": "?data_type",
        "type": "data_types"
      }
    ],
    "preconditions": [
      "has_data(database, ?user, ?data_type)",
      "is(?actor, marketing_manager)"
    ],
    "effects_add": [
      "has_data(?actor, ?user, ?data_type)"
    ],
    "effects_del": [],
    "justifications": [
      "None of the information collected is sold or shared with any third party and is available only to specific Marketing Managers at Mohegan Sun."
    ],
    "schema_key": "access_user_data|marketing_manager|user,data_types"
  },
  {
    "name": "signup_momentum_member",
    "actor": "user",
    "parameters": [
      {
        "var": "?user",
```

```
          "type": "user"
        }
      ],
      "preconditions": [
        "at_location(?user, players_club_booth)"
      ],
      "effects_add": [
        "is_momentum_member(?user)",
        "has_data(company, ?user, name)",
        "has_data(company, ?user, home_address)",
        "has_data(company, ?user, phone_number)",
        "has_data(company, ?user, email_address)",
        "has_data(database, ?user, name)",
        "has_data(database, ?user, home_address)",
        "has_data(database, ?user, phone_number)",
        "has_data(database, ?user, email_address)"
      ],
      "effects_del": [],
      "justifications": [
        "In addition to the information collected online
           , patrons may sign up to become a Momentum
           member upon visiting Mohegan Sun. All personal
            information collected at any Player's Club
           booth including name, address, phone number
           and email address is placed in a database that
            is not shared or sold with any third party."
      ],
      "schema_key": "signup_momentum_member|user|user"
    },
    {
      "name": "unsubscribe_from_emails",
      "actor": "user",
      "parameters": [
        {
          "var": "?user",
```

```
          "type": "user"
        }
      ],
      "preconditions": [
        "subscribed_to_emails(?user)"
      ],
      "effects_add": [],
      "effects_del": [
        "subscribed_to_emails(?user)"
      ],
      "justifications": [
        "Anyone receiving email updates from Mohegan Sun that wish to be removed should immediately click on the \"unsubscribe\" link which is included in every email correspondence that is transmitted from Mohegan Sun."
      ],
      "schema_key": "unsubscribe_from_emails|user|user"
    },
    {
      "name": "update_user_information",
      "actor": "customer_service_rep",
      "parameters": [
        {
          "var": "?user",
          "type": "user"
        },
        {
          "var": "?data_type",
          "type": "data_types"
        }
      ],
      "preconditions": [
        "is(?actor, customer_service_rep)",
        "has_data(database, ?user, ?data_type)"
```

```
      ],
      "effects_add": [
        "has_data(database, ?user, ?data_type)"
      ],
      "effects_del": [],
      "justifications": [
        "Certain information, such as address or email address, can be updated or corrected simply by calling Customer Service at 1.888.777.7920."
      ],
      "schema_key": "update_user_information|customer_service_rep|user,data_types"
    },
    {
      "name": "verify_age_for_promotion",
      "actor": "company",
      "parameters": [
        {
          "var": "?user",
          "type": "user"
        }
      ],
      "preconditions": [],
      "effects_add": [
        "is_21_or_older(?user)"
      ],
      "effects_del": [],
      "justifications": [
        "All recipients of any online promotions must be 21 years of age or older. Age is verified upon contact of the specified winner."
      ],
      "schema_key": "verify_age_for_promotion|company|user"
    }
```

```
  ]
}
```

A.1.3 Reachability checker algorithm

Listing A.3: Reachability check

```
send_email_broadcast: requested_emails(?user)
signup_momentum_member: at_location(?user, players_club_booth)
access_user_data: is(?actor, marketing_manager)
update_user_information: is(?actor, customer_service_rep)
```

A.1.4 Latent action generator

Listing A.4: Latent action generator (Sonnet 4)


```
{
  "latent_actions": [
    {
      "name": "request_email_subscription",
      "actor": "user",
      "parameters": [
        {
          "var": "?user",
          "type": "user"
        }
      ],
      "preconditions": [],
      "effects_add": [
        "requested_emails(?user)",
        "subscribed_to_emails(?user)"
      ],
      "effects_del": [],
      "supports_action": [
        "send_email_broadcast"
```

```
      ],
      "satisfies_fluents": [
        "requested_emails(?user)"
      ],
      "rationale": "Users must be able to request email subscriptions to receive broadcasts. This action captures the initial opt-in process.",
      "justifications": [
        "latent_generated"
      ],
      "schema_key": "request_email_subscription|user|user"
    },
    {
      "name": "visit_players_club_booth",
      "actor": "user",
      "parameters": [
        {
          "var": "?user",
          "type": "user"
        }
      ],
      "preconditions": [],
      "effects_add": [
        "at_location(?user, players_club_booth)"
      ],
      "effects_del": [],
      "supports_action": [
        "signup_momentum_member"
      ],
      "satisfies_fluents": [
        "at_location(?user, players_club_booth)"
      ],
      "rationale": "Users must physically visit the Player's Club booth to sign up for Momentum
```

```
        membership. This represents the physical action of going to that location.",
      "justifications": [
        "latent_generated"
      ],
      "schema_key": "visit_players_club_booth|user|user"
    }
  ]
}
```

A.2 Small model outputs

Outputs from experimentation with smaller models.

A.2.1 Global context extractor

Listing A.5: Global context agent (Haiku 3.5)

```
{
  "definitions": {
    "personal_information": ["PI", "Personal Information", "user data"],
    "momentum_member": ["Player's Club member"],
    "website": ["site", "mohegansun.com"]
  },
  "global_defaults": [
    {
      "rule": "Company does not sell or share personal information with third parties",
      "source_spans": [
        "Mohegan Sun does not share or sell your personal information to anyone",
        "The database is available for Mohegan Sun's usage only and is not shared with anyone"
      ]
```

```
    },
    {
      "rule": "Company does not participate in SPAM",
      "source_spans": ["Mohegan Sun does not participate
          in any format of SPAM and will not send any
         unsolicited emails"]
    },
    {
      "rule": "Age restriction for online promotions",
      "source_spans": ["All recipients of any online
         promotions must be 21 years of age or older"]
    }
  ],
  "entities": {
    "actor": {
      "company": {
        "name": "Mohegan Sun",
        "type": "Casino and online service provider",
        "primary_contact": {
          "phone": "1.888.226.7711",
          "email": "information@mohegansun.com",
          "address": "1 Mohegan Sun Boulevard Uncasville
             , CT 06382"
        }
      },
      "user": {
        "type": "Website visitor or casino patron",
        "potential_roles": ["Momentum member", "Online
           visitor", "Potential promotion recipient"]
      }
    },
    "system": {
      "database": {
        "description": "Internal storage for user
           information",
```

```
        "access_limited_to": "Specific Marketing Managers at Mohegan Sun"
      },
      "website": {
        "domain": "mohegansun.com",
        "age_restriction": "Not intended for children"
      }
    },
    "data_type": {
      "personal_information": {
        "fields": [
          "name",
          "email address",
          "home address",
          "phone number",
          "areas of interest"
        ],
        "collection_points": ["Online", "Player's Club booth"]
      }
    },
    "purpose": {
      "data_collection": {
        "primary": "Email broadcasts with tailored information",
        "secondary": "Momentum membership management"
      }
    },
    "policy_feature": {
      "data_management": {
        "update_method": "Call Customer Service at 1.888.777.7920",
        "unsubscribe_method": "Click 'unsubscribe' link in email"
      },
```

```
      "child_protection": {
        "policy": "No data collection from children",
        "supervision_requirement": "Direct supervision by parent or legal guardian"
      }
    }
  }
}
```

A.2.2 Action extraction loop

Listing A.6: Action extraction loop (Haiku 3.5)

```
{
  "entities_delta": {
    "roles": {
      "marketing_manager": "Specific staff member with access to user information",
      "customer_service_rep": "Staff member who can update user information",
      "user": "Individual providing personal information",
      "parent_guardian": "Supervising adult for underage users"
    },
    "resources": {
      "personal_data": "Information collected about users",
      "email_system": "System for sending communications",
      "database": "Storage system for user information"
    }
  },
  "fluents_delta": [
    "has_data(?data_holder, ?data_owner, ?data_type)",
```

```
    "is(?entity, ?role)",
    "age_verified(?user, ?age)",
    "subscribed(?user, ?communication_type)"
  ],
  "actions": [
    {
      "name": "collect_user_data",
      "actor": "user",
      "parameters": [
        {"var": "?user", "type": "user"},
        {"var": "?dt", "type": "personal_data"}
      ],
      "preconditions": [
        "age_verified(?user, 21)"
      ],
      "effects_add": [
        "has_data(mohegan_sun, ?user, ?dt)"
      ],
      "justifications": [
        "\"Information collected online is used for email broadcasts utilizing tailored information upon the request of a visitor.\""
      ],
      "schema_key": "collect_user_data|user|[user, personal_data]"
    },
    {
      "name": "update_user_information",
      "actor": "customer_service_rep",
      "parameters": [
        {"var": "?user", "type": "user"},
        {"var": "?dt", "type": "personal_data"}
      ],
      "preconditions": [
        "has_data(mohegan_sun, ?user, ?dt)"
```

```
      ],
      "effects_add": [
        "has_data(mohegan_sun, ?user, ?dt)"
      ],
      "justifications": [
        "\"Certain information, such as address or email address, can be updated or corrected simply by calling Customer Service at 1.888.777.7920.\""
      ],
      "schema_key": "update_user_information|customer_service_rep|[user,personal_data]"
    },
    {
      "name": "unsubscribe_email",
      "actor": "user",
      "parameters": [
        {"var": "?user", "type": "user"}
      ],
      "preconditions": [
        "subscribed(?user, email_updates)"
      ],
      "effects_del": [
        "subscribed(?user, email_updates)"
      ],
      "justifications": [
        "\"Anyone receiving email updates from Mohegan Sun that wish to be removed should immediately click on the 'unsubscribe' link which is included in every email correspondence\""
      ],
      "schema_key": "unsubscribe_email|user|[user]"
    },
    {
      "name": "remove_child_data",
```

```
        "actor": "parent_guardian",
        "parameters": [
          {"var": "?child", "type": "user"}
        ],
        "preconditions": [
          "has_data(mohegan_sun, ?child, ?dt)",
          "is(?child, under_21)"
        ],
        "effects_del": [
          "has_data(mohegan_sun, ?child, ?dt)"
        ],
        "justifications": [
          "\"Children may use this site only with the direct supervision of their parent or legal guardian... Any parents or guardians who are aware of children submitting information should notify us at once of the specific information so that we may remove them from any mailing lists.\""
        ],
        "schema_key": "remove_child_data|parent_guardian|[user]"
      }
    ]
}
```

APPENDIX

B

DOMAIN ENCODING PROMPTS

B.1 Agent prompts

B.1.1 Definitions

These are passed to agents if relevant to their task:

Closed world assumption: If an action/operator is not present in the final domain, it is considered forbidden. If a fact is not in the state, it is false. Forbidden actions can be left out of the domain.

Fluent: A predicate like `has_data(holder, user, email)` or with variables like `has_data(holder, user, ?dt)`. All notions of posessing information should be normalized to the fluent `has_data(?data_holder, ?data_owner, ?data_type)`. All role checks should be normalized to `is(?entity, ?role)`, like `is(?actor, lawyer)`. `?actor` is a special variable to refer to the actor of an action.

Variable/Constant distinction: In every fluent argument, either supply a variable (e.g., `?dt`) and declare its type in `parameters`, or supply a constant (e.g., `email`). Never leave a slot ambiguous. `?actor` is a special variable to refer to the actor of an action.

Guarded event: A real-world action/event that includes role or resource constraints as preconditions. For example "only a company lawyer is permitted to access user data" should become an `access_data` action, with the preconditions that the company possesses said data, and the actor is a lawyer; and the effect that the lawyer now has said data.

B.1.2 Global context extractor

You are building global context for a privacy policy extraction pipeline. Read the entire policy text and produce a structured context that downstream agents can rely on. When you reference policy content, include minimal quotes or location spans under source_spans.

Tasks:

1. definitions: Map canonical terms to aliases, e.g., "personal_information": ["PI", "Personal Information"].
2. global_defaults: High-level rules that apply across the policy (e.g., “We do not sell personal information”).
3. List of all entities mentioned in the text. Classify them as actor, system, data_type, purpose, etc. Include short definition. Always include two actors: 1) "company" as the entity whose privacy policy is being processed (unify explicitly mentioned company name with "company"). 2) User

Output strict JSON with schema:

```
{
  "definitions": { "string (snake\_case)": ["string", "..."] },
  "global\_defaults": [ { "rule": "string", "source\_spans": ["string"] } ],
  "entities": {
    "string (category, snake\_case)": {
        "string (category, snake\_case)": "string (definition)",
```

```
            ...
        },
        ...
    }
}
```

B.1.3 Action extraction pass

You are extracting only explicitly permitted or required actions from the given policy text, to be part of a narrative planning domain. Treat permission wording as guarded events with preconditions, not as "grant/allow" meta-actions. Enforce variable vs constant marking and resource-use -> precondition (if the action uses data or another resource, require it in preconditions). Merge multiple sentences that describe the same real-world event into one action with the union of effects. Try to use entities from the attached global context.

For each action, include:

1. name (snake_case), actor, and typed parameters (each variable starts with ?).
2. preconditions and effects_add/effects_del, with constants as bare symbols (e.g., 'email') and variables like ?dt. Remove any preconditions that are always true according to the policy text.
3. justifications with quotes from original text
4. schema_key as <name>|<actor>|<parameter_types> to help de-duplication

Output strict JSON with schema:

```
{
    "entities_delta": {
        "string (category, snake_case)": {
            "string (category, snake_case)": "string (definition)",
            ...
        },
        ...
```

```
  },
  "fluents_delta": ["fluent_schema"],
  "actions": [
    {
      "name": "string",
      "actor": "string",
      "parameters": [ { "var": "?x", "type": "
        string" } ],
      "preconditions": ["fluent_or_expression"],
      "effects_add": ["fluent_or_expression"],
      "effects_del": ["fluent_or_expression"],
      "justifications": [ "quote", ... ],
      "schema_key": "string",
    }
  ]
}
```

B.1.4 Latent action generation pass

Given a narrative planning domain based on a privacy policy, along with a set of unreachable actions (unsatisfiable preconditions), propose a minimal set of actions that produce the missing fluents (preconditions). Preserve var/const distinction. Do not extrapolate infinitely ("user_born", etc.). Limit maximum causal depth from given actions to 3.

Output strict JSON with schema:

```
{
  "latent_actions": [
    {
      "name": "string",
      "actor": "string",
      "parameters": [ { "var": "?x", "type": "
        string" } ],
      "preconditions": ["fluent_or_expression"],
      "effects_add": ["fluent_or_expression"],
```

```
        "effects_del": ["fluent_or_expression"],
        "supports_action": ["explicit_action_name"],
        "satisfies_fluents": ["fluent_schema"],
        "rationale": "string",
        "justifications": ["latent_generated"],
        "schema_key": "string"
    }
  ]
}
```

B.1.5 Domain fixer

Merge all explicit and latent actions into a consistent domain:

1. Normalize symbols using definitions in context
2. Deduplicate by schema_key
3. union effects/preconditions only when justified by text or rationale
4. Ensure every fluent appearing in actions exists in the fluents list

Output strict JSON with schema:

```
{
  "entities": {
    "string (category, snake_case)": {
        "string (category, snake_case)": "string (definition)",
        ...
    },
    ...
  },
  "fluents": ["fluent_schema"],
  "actions": [
    {
      "name": "string",
```

```
        "actor": "string",
        "parameters": [ { "var": "?x", "type": "string" } ],
        "preconditions": ["fluent_or_expression"],
        "effects_add": ["fluent_or_expression"],
        "effects_del": ["fluent_or_expression"],
        "justifications": [ "quote", ... ],
        "schema_key": "string",
    }
  ]
}
```