Query-Based Database Policy Assurance Using Semantic Web Technologies

by

José Hiram Soltren

S.B., Massachusetts Institute of Technology (2007)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

© Massachusetts Institute of Technology 2009. All rights reserved.

Certified by

Lalana Kagal Research Scientist Thesis Supervisor

Accepted by Dr. Christopher J. Terman Chairman, Department Committee on Graduate Theses

Query-Based Database Policy Assurance Using Semantic Web Technologies

by

José Hiram Soltren

Submitted to the Department of Electrical Engineering and Computer Science on August 21, 2009, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we present a novel approach to database security which looks at queries instead of the database tables themselves. In doing so, we use a number of Semantic Web technologies to define rules, translate queries, and make assertions about compliance with existing policies. We can ascertain compliance without looking at the contents of the database. Our system can function as a drop-in addition to an existing database system, adding additional functionality in a robust manner. The policies are written in the AIR language, and the reasoners and wrapper scripts in C++ and Python. We discuss the design and implementation of this system in detail.

Thesis Supervisor: Lalana Kagal Title: Research Scientist

Acknowledgments

Prior to beginning work on this thesis, Semantic Web technologies were entirely new to me. Learning all of the technologies has been a fun and enlightening journey.

My thanks go to everyone who helped to make this work possible. Lalana Kagal played an instrumental role as my advisor, a sounding board for many ideas and a source of much help. The other graduate students here in DIG, especially Oshani Seneviratne, Ian Jacobi, and Jim Hollenbach, provided much useful feedback from the very beginning. I am equally indebted to our summer UROP interns, especially Yotam Aron, for providing much needed help with this project and helping to keep a fresh perspective.

I am also indebted to Prof. Hal Abelson and Danny Weitzner. Their undergraduate course, Ethics and Law on the Electronic Frontier (6.805), was my introduction to the world of policy in the digital domain.

Everyone who read my thesis and provided valuable feedback merits special recognition. Lalana Kagal and Hal Abelson, as well as Ray Li, K. Krasnow Waterman and Joe Pato, have, between all of them, well over a century of experience in Web, software, legal, and security systems. My thanks to them for making this thesis cohesive.

Finally, I thank my girlfriend, Jane, for her patience and understanding as this thesis approached completion, and my mother, Diana, who supported my decision to come to MIT from the very beginning.

This work was supported by an IARPA Award, #FA8750-07-2-0031.

Contents

1	Intr	roduction	15				
	1.1	Motivating Example	17				
		1.1.1 Sample Usage Scenario	17				
	1.2	System Components	18				
	1.3	Outline	19				
2	Poli	icy Assurance	21				
	2.1	Introduction to Policy Assurance	21				
	2.2	User Roles and Perspectives	23				
		2.2.1 The Administrator	23				
		2.2.2 The User	24				
		2.2.3 The Auditor	24				
	2.3	Modes of Operation	25				
	2.4	4 Demonstration					
		2.4.1 Describing a Free-Text Policy	25				
		2.4.2 Checking a Compliant Query	26				
		2.4.3 Checking an Incompliant Query	31				
		2.4.4 Demo Notes	37				
	2.5	Summary	37				
3	Sys	tem Detail	39				
	3.1	SPARQL Query Translation	39				
		3.1.1 SPARQL to N3 Web Page	41				

		3.1.2	Query Conversion Ontology	41
		3.1.3	swobjects: Parsing and Serializing	43
		3.1.4	SPARQL Language Translation	43
		3.1.5	Lost in Translation	52
		3.1.6	Translator Summary	54
	3.2	AIR F	Policy Generation	54
		3.2.1	Templates for Policy Generation	54
		3.2.2	Supported Policy Types	54
		3.2.3	Automatic Policy Generation	63
		3.2.4	Query History with check-compliance	67
		3.2.5	Policy Generation User Interface	68
		3.2.6	Compliance Testing and Browser Presentation in Tabulator	68
		3.2.7	Implementation Note	68
	3.3	Summ	nary	72
4	Per	formai	nce	73
5	Rel	ated a	nd Prior Work	77
	5.1	Policy	Awareness	77
	5.2	Metho	odologies of Access Control	78
		5.2.1	Mandatory and Discretionary Access Control	78
		5.2.2	Role Based Access Control	79
		5.2.3	Rule- and Policy-Based Access Control	80
	5.3	Prior	Work in Relational Databases	80
		5.3.1	Access Control Lists	80
		5.3.2	Access Control Features In A Modern RDBMS	81
		5.3.3	Misuse and Intrusion Detection	82
	5.4	Altera	tion of Data	83
6	Fut	ure Di	rections	85
	6.1	SPAR	QL Endpoint Integration	85

	6.2	SQL Support	86
	6.3	Completing and Porting the N3 Translator	86
	6.4	Policy Generation from Natural Language	87
	6.5	Semantic Policies	87
	6.6	Database Description	87
7	Con	cluding Thoughts	89
A	Bac	kground Technologies	91
	A.1	Semantic Web Overview	91
		A.1.1 The Vision	91
		A.1.2 The URI	93
		A.1.3 HTML, the HyperText Markup Language, and XML, the eX-	
		tensible Markup Language	94
	A.2	RDF	95
		A.2.1 Notation 3	97
	A.3	OWL	97
	A.4	Tabulator	98
	A.5	SPARQL	98
	A.6	Reasoning	100
		A.6.1 Forward Chaining	100
		A.6.2 Production Rule Systems	101
		A.6.3 The Rete Algorithm	101
		A.6.4 Semantic Web Application Platform	101
		A.6.5 cwm and cwmrete	101
	A.7	AIR	102
		A.7.1 Introduction	102
		A.7.2 A Brief AIR Tutorial	103
		A.7.3 Changes to the AIR language	106
		A.7.4 AIR Summary	108
	A.8	Summary	108

В	Sup	porting Code	109
	B.1	MIT Prox Card Policy	109
	B.2	SSN Policy - Original Ontology	110
	B.3	SSN Policy - Current Ontology	115
	B.4	A sample SPARQL Query	117
	B.5	Abstract SPARQL to N3 Ontology	117
	B.6	Sample Restriction Policy	119
	B.7	Sample Inclusion Policy	121
	B.8	Sample Exclusion Policy	124
	B.9	Sample History-Aware Exclusion Policy	125
	B.10	Sample Chaining Policy	127
	B.11	Sample Default Deny Policy	129
	B.12	No-Address Restriction Policy for Sample Scenario	130

List of Figures

1-1	Architecture of our policy assurance reasoner, demonstrating separa-	
	tion from the RDBMS	19
2-1	Overview of an integrated policy assurance system. From [1]	23
2-2	Restriction Policy Creator Web Page	27
2-3	Tabulator Representation of the Sample "No Address" Restriction Policy	28
2-4	URI for the demo "no address" policy.	28
2-5	Converting a SPARQL Query to N3	29
2-6	Compliant SPARQL query, converted to N3	30
2-7	Tabulator Representation of a Sample SPARQL Query	30
2-8	URI for the translation of the compliant demo query.	31
2-9	Policy Execute Web Page	32
2-10	Tabulator Compliance Summary for Sample Query and Sample Policy	33
2-11	Tabulator Compliance Justification UI for Sample Query and Sample	
	Policy	34
2-12	Tabulator Non-Compliance Summary for Sample Query and Sample	
	Policy	35
2-13	Tabulator Non-Compliance Justification UI for Sample Query and Sam-	
	ple Policy	36
2-14	URI for the translation of the non-compliant demo query	36
3-1	A screenshot of the SPARQL to N3 translator Web page	40
3-2	Ontology diagram of the SPARQL translation, courtesy of Yotam Aron.	42
3-3	A screenshot of the policy generator, courtest of Yotam Aron	69

3-4	Tabulator browser presentation of the MIT Prox Card policy	70
3-5	Tabulator justification user interface.	71
4-1	Unoptimized policy run time from table 4.1.	75
4-2	Unoptimized policy run time from table 4.2. Note log scale on the X	
	axis	76
A-1	Semantic Web "layer cake," showing how components relate [2]	92
A-2	The RDF logo. Its structure suggests the triple pattern. From [3]	95
A-3	A sample RDF data set, using backslash to denote newline. From [4].	99
A-4	A sample SPARQL query. From [4]	99
A-5	Diagram of the AIR policy language ontology.	103

List of Tables

Query conversion ontology.	41
Conversion of Boolean operators. The "name" refers to the title of	
the operator in the translator code. The "translation" is what the	
translator outputs, in the query translation namespace	50
Automatic policy generation: USE	65
Automatic policy generation: RETRIEVE	66
Running time, in seconds, of unoptimized queries	74
Running time, in seconds, of optimized queries.	75
Expressing natural language as triples.	95
Expressing natural language as triples.	96
The output of a basic SPARQL query.	99
	Query conversion ontology

Chapter 1

Introduction

Databases are a common way of storing data, and SQL is arguably the most popular variety of database in production as of this writing. The most popular SQL databases implementations, Oracle and MySQL, offer a rudimentary set of security options. SQL has a table-centric view of data, and a UNIX-like security model. [5] A database super-user can create individual tables, create user accounts, and grant or deny access to certain tables to certain users.

The existing state of the art in database security is administrator-centric and based on security models that are decades old. While adequate for many applications, these systems have limitations. They provide an "all or nothing" model of security, with no ability to show which rules provided an output or a specialized, data centric policy system. More information about the conclusions that a policy system reached is helpful. It promotes transparency, helps find errors, helps keep users honest, and helps to promote more fine-grained access policies. I expand on these systems with the use of Semantic Web technologies. In particular, I offer policy assurance, which provides a guarantee that a database query was or was not in compliance with a policy. Policy assurance provides an exact description of why a query is or is not compliant.

To securely grant access to a piece of information, data security engineers typically ask the following questions, or make assumptions as necessary:

- Who are you?
- What information are you trying to see?
- Why do you want to see this information?

Electronic systems traditionally utilize a few common design patterns in their implementations of security primitives. Discretionary access control [6] is the most familiar approach, and the basis of the UNIX security model. A user or super-user can expressly allow or deny other users certain permissions to certain pieces of data. Mandatory access control [7] requires explicit permission for any action, granted by an outside body; for a time, this was a popular design pattern for government systems containing classified data. Role-based access control [8] groups users into specific roles, which themselves have specific permissions. Rule based access control [9] allows an administrator to create more specific policies for governing access.

This thesis documents a database security system built using Semantic Web technologies and design principles. I argue that a query to a database contains a substantial amount of information about what a user is trying to access, and why they wish to do so. I design and implement a system that implements rudimentary database security, and provides explanations for the policies it enforces. Our model user is "honest but curious": a trusted, well-intentioned employee of a government agency who is using a database, provided under numerous policies and agreements, to the user's agency.

The thesis introduces some key contributions to the field. The SPARQL to N3 conversion tool and related ontology are novel. The definitions of template policies, and tools for creating them, reduce policy based security to a few primitives. Policy assurance provides more transparency to users and administrators alike.

I approached this project in two phases. In phase one, I used SWObjects to convert SPARQL queries to an RDF serialization. I wrote low level AIR policies to check the compliance of these SPARQL queries, and demonstrated some test cases. In phase two, I created an abstraction for these SPARQL queries, and provided a meta level that removed dependence on the data structure.

1.1 Motivating Example

"Policy Assurance" is a set of technical mechanisms that enable effective and accountable information sharing and usage. [10]

"Policy assurance" is a process by which we can be anywhere from reasonably to absolutely certain that actions in a system are in compliance with an existing policy. To validate the claim of compliance (or noncompliance), there is some data set which validates the claim.

As a sample scenario, we consider a database that contains highly sensitive data and, as a result, is only accessible to a limited number of people. For this possible database, we assume that the database administrator is not the database owner, and thus, cannot see either the contents of the database or the queries that external parties make to the database. The database owner may only see the information in the database, but not the queries made to it. Policy assurance allows the database owner and database administrator to verify that every query made to the database was in compliance with the given policies, without exposing any sensitive information.

The key point of policy assurance is that, if an external party knows the policies and has access to enough information about each policy check, the external party can verify the compliance of the policies. If there is a guarantee that every single access is logged, then an external party can validate a claim of complete compliance. This promotes transparency, by allowing verification, while providing security, which we might define as "minimize the information required to permit verification."

1.1.1 Sample Usage Scenario

In this thesis, we will consider a simple usage scenario of an administrator who wishes to guard data in an RDF data set which users access using SPARQL queries. In this scenario, users may not access location information about entities in the datbase; specifically, they may not retrieve or use address information in any form. The administrator may not see the queries that users are making, for security reasons, but must be certain that *every single query* made to the data set is compliant with the policy about addresses.

The system described in this thesis is well suited to this task. In later sections, we will describe how the administrator would create a policy to enforce this, and how the administrator would check some sample queries manually. We will then describe how this system could be configured to perform this task automatically, providing policy assurance for the administrator and security for the users.

1.2 System Components

In this thesis, I present a system that provides policy assurance for SPARQL databases by checking whether incoming queries conform to authorization policies written in AIR. The system uses query based security to draw its conclusions. This system has several components:

- A translator, which generates an N3 output from a SPARQL query. This output serves as an input to a reasoner.
- An ontology which serves to define the N3 output of the translator.
- A number of pre-defined policy templates in AIR, which capture the behavior of some of the most common policy design patterns. Some policies support reasoning over a log of previous queries.
- An algorithm for automatically generating AIR policies by combining user inputs with policy templates.
- A Web interface for performing SPARQL to N3 translation.
- A Web interface for generating policies.
- A Web interface for checking queries against policies.

All of the Web interfaces in this thesis work with Tabulator's Justification UI, for user-friendly viewing of queries and policies.



Figure 1-1: Architecture of our policy assurance reasoner, demonstrating separation from the RDBMS.

1.3 Outline

The remainder of this thesis is structured as follows. The following chapter provides a demonstration of the system, elaborates on the topic of policy assurance, and provides an overview of how this system is useful in implementing policy assurance. Following the overview, I discuss the design and implementation of each component of the system in detail, with a discussion of the design assumptions and trade-offs necessary for implementation. Then, I explore the performance of the reasoner. I then look at future directions for this work, and draw comparisons with other work in the field, before coming to a conclusion.

The appendix chapters provide background information. I discuss the ongoing Semantic Web initiative and the technologies which provide the infrastructure for this work. I offer code samples and useful information about the project code.

Chapter 2

Policy Assurance

2.1 Introduction to Policy Assurance

In a broad sense, policies exist in almost any situation where multiple agents are in contention for a shared resource. If these agents were in cooperation, or if there were sufficient resources, there would be no contention, and thus, no need for a policy system. The policies give a system a basis for mitigating contention.

Policies manifest themselves in multiple forms: as laws and customs in a human world; as permissions and restrictions in a digital world; as habits in nature. Policy seeks to identify particular actions, perhaps evaluated in a particular context, and make a decision of some sort as a result of those actions. Policies exist for a number of reasons. They may protect a shared resource from overuse or from simultaneous users, which may damage the resource. They may regulate or restrict access to certain kinds of resources, as access may cause side effects within a system. Policies themselves often have justifications, to explain their existence.

In the everyday world, rules may exist for our own protection, or the protection of others. As an example, I consider the public road system. There are a large number of policies which concern the use of public roads, for any purpose, by any individual. The existence of the policies is something of a fair trade bargain: without the policies, the road system would deteriorate to the point of being useless; without a road system, many other activities (such as trade, visiting others, providing medical care, obtaining food, etc.) would become exceedingly difficult. The policies may not be ideal, but are a superior option to the complete absence of the system.

A well-known policy states that "all users of motorized vehicles on public roads must be certified." In common terms, I know this as *all drivers must have a license*. This is a simple, if expensive, policy to enforce: an agent (usually a police officer) may check for compliance of this policy by asking someone operating a motorized vehicle to present their license for inspection. This policy is expensive to enforce because the means for doing so involve either having a police officer check everyone's license (which is not feasible), or requiring a license in order to enable a vehicle's operation (which may also be infeasible). As a result, there is some non-zero probability that there are unlicensed drivers on the road.

In this case, complete policy assurance is exceedingly difficult. Every time an officer checks a driver's license, there is a record which includes the date and time of the check, the officer's identity, the identity of the driver and the vehicle, and information about the driver's license. This provides policy assurance for a single incident, but complete assurance is not feasible. In practice, the choice to check every single action versus some probabilistic number of interactions is itself a policy decision, hopefully the result of a rigorous risk or cost-benefit analysis.

In a digital world, the marginal cost of explicit checks and storing lots of data are far lower, and in many situations, effectively negligible or even zero. Thus, it is easily possible to perform a rigorous check of every action against policies. As an example, consider a Web site with a policy that states, "all users of the advanced features of this Web site must have a valid account." Regardless of the definitions of user, feature, valid, and account that a Web site may choose, it is straightforward to perform a check every time a user tries to access an advanced feature. The Web site may maintain a log of successful and unsuccessful attempts to use advanced features.

We believe that the AIR language, reasoner, and Semantic Web approach is a good fit for the problem of securing a database with sensitive data [1]. The policies we write only need to look at the queries that a user makes. In the case of most policies, we only need to look at a single query to determine compliance. The reasoner looks at



Figure 2-1: Overview of an integrated policy assurance system. From [1].

a user's query, and an administrator's policy, compares the two, and finds code in the query that matches some template in the policy. The reasoner output provides positive, verified confirmation that a query is or is not in compliance with a policy, while only divulging enough information from the query to back up the compliance claim. The policy assurance approach fulfills the need for positive confirmation and traceability of query compliance using the minimal amount of data. Indeed, if the very contents or results of the query are confidential, we must tread carefully.

2.2 User Roles and Perspectives

The system we present herein has multiple usage scenarios, to help end users implement this system and integrate it into existing databases.

2.2.1 The Administrator

A database administrator, or DBA, would be the first user to interact with the system. The DBA is the entity responsible for maintaining a data set, and thus, for creating policies that regulate access to the data set. It is possible that the DBA does not have access to the queries that a user will make, and it is also possible that the DBA only has access to a data set's metadata.

In order to create a policy, a DBA must have a list of the fields in a database, and

in particular, the data types of those fields as URIs. Our implementation of policies at present is largely dependent on finding a query that binds to a particular data type. I are working on a tool that helps to automate this process by determining what types are used in a data set, though this is moot if the DBA cannot access the data set proper.

It is up to the DBA to determine what kinds of policies they wish to implement. The DBA may be bound by local and national laws, by department practices, or by any number of other factors in creating policies. In all likelihood, the policies that the DBA needs to implement will be expressible in terms of the primitives that I define in the next chapter. The DBA would then use the Web based tool to create policies using our templates, and possibly check for compliance using some sample queries the same way a user would. Some policies can be "history aware", meaning that the policy looks at all of a user's past queries in addition to the current query when making a compliance decision.

2.2.2 The User

The second major user of this system is someone who wishes to access the database. There are two possible modes of operation here. If the DBA or a system administrator has configured the policy assurance system as a SPARQL add-in, it is possible that the user will see no change, other than having some queries rejected for lack of compliance. Holver, if the DBA chooses to implement the Web-based option, the user would be able to see the compliance output of their query in a Tabulator-equipped Web browser. The reasoner's output would be helpful in aiding the "honest but curious" user to make compliant queries to the database.

2.2.3 The Auditor

The third user of the policy assurance system is the auditor. This is a person or entity charged with the responsibility of assuring that the policies that the DBA wrote are correct, and that the system achieves compliance. The auditor would be able to access the query history, the policies, and the reasoner outputs, and manually verify that things are working correctly. An important future work of this project is to provide tools that help the auditor perform query analysis.

2.3 Modes of Operation

As implemented in this thesis, the policy assurance system exists entirely outside of any database implementation. It would allow analysis of a query history by a user or administrator, and allow a user or administrator to check new queries for compliance before sending them to the database. With no further modification, this system could work, slowly but effectively, in a hypothetical "air gap" environment where a user sends a query to an administrator for manual verification and entry. An important future work is to integrate this completely with a SPARQL endpoint; we describe the work needed here in the Future Work section.

2.4 Demonstration

In this section, we demonstrate the workflow for an administrator who wishes to create a policy and use it to check queries for compliance using the Firefox Web browser with the Tabulator browser plugin. This section expands on the sample use scenario presented in the introduction. The workflow for all policy types is similar.

2.4.1 Describing a Free-Text Policy

The administrator wants to encode a policy that says, "users may not find out where members of the database live. In the case of my data set, this means that users may not see any data of the type ex:address. Users may not USE or RETRIEVE such data."

The first thing that the administrator needs to do is to create a policy. The administrator heads to the policy generation page,

http://dig.csail.mit.edu/2009/policy-assurance/generator/

The administrator clicks the "Restriction Policy" link, since this policy is most in line with what the administrator wants to do. The administrator enters the following input:

- Policy name: no-address
- Policy description: Users may not find the home address of members of the database.
- Namespaces: @prefix ex: <http://example.com/#>.
- Included attributes: Variable: ex:address. Click "both" to check USE and RETRIEVE.

The administrator clicks "Submit!" to generate the policy (see figure 2-2), and it appears in Tabulator as shown in figure 2-3.

The administrator can type Ctrl+U to see the source AIR code of the policy. The source of this policy is in the appendix B.12.

The policy has a unique URI. Since the administrator will need the URI later, the administrator saves the URI, as in figure 2-4.

2.4.2 Checking a Compliant Query

An administrator wants to check the following SPARQL query for compliance. Since it does not mention ex:address, the administrator suspects that it will be compliant:

```
PREFIX example: <http://example.com/#>
```

```
SELECT * WHERE {
    ?s example:ssn ?ssn.
    ?s example:age ?age.
    ?s example:name ?name.
    FILTER (?age > 18)
```

}

Policy-genera	tor UI - Mozilla Firefox 📃 🗖 🔀
<u>File E</u> dit <u>V</u> iew	Hi <u>s</u> tory <u>B</u> ookmarks <u>T</u> ools <u>H</u> elp
S> - C	🗙 🏠 🗋 http://dig.csail.mit.edu/2009/policy-assuran: 🏠 🔹 🛃 Google 🖉
Policy-gener	ator UI +
Policy	Restriction Policy Creator
Type List	A Restriction Policy does not allow a person to use, retrieve, or both use and retrieve a variable with a certain attribute.
	Policy Name (Optional)
<u>Restriction</u> <u>Policy</u>	Please specify a policy name if desired.
 Inclusion Policy 	no-address
 <u>Exclusion</u> <u>Policy</u> 	Policy Description (Optional)
• <u>Default Deny</u> <u>Policy</u> • <u>Chaining</u> <u>Policy</u>	Users may not find the home address of members of the database.
	Namespaces Please provide the namepaces you wish to be used in the policy. Place each URI on its own line. Make sure to separate them with commas, and provide a unique prefix for each. @prefix ex: ">http://example.com/#> Descent the attributes Included Attributes Please list the attributes that may not be included, and whether for retrieval, use, or both. Please remember to write the attributes using the correct prefix Variable: ex:address O Use Retrieve
Policy-gene Policy Type List • Restriction Policy • Exclusion Policy • Exclusion Policy • Original Policy • Chaining Policy	Add Another Variable
Done	Submit!

Figure 2-2: Restriction Policy Creator Web Page

	C X G	「 」 http://dig.csail.mit.edu/2009/policy-assur; 公 ・ 【 るoogle し
Policy-gener	ator UI	🔄 📄 http://dig.csail.mit.edu/2009/pol🛛 🔶
nttp://dig.csail.n xtDescription=U xample.com/#>	nit.edu/2009/polic Jsers+may+not+fi >.&Attribute1=ex:a	;y-assurance/generator/make_restriction_policy.py?policyName=no-address& nd+the+home+address+of+members+of+the+database.&namespaces=@prefix+ex:+ <http: address&Var1=both&url=whatever</http:
o address	label rule	Users may not find the home address of members of the database. no address check
	type description label	Policy Make sure this is a query. no-address query check
o address check	pattern	clause W Q retrieve P type SPARQLQuery
	rule type alt	no address rule 0 Belief Rule rule no address rule 1
	assert	Q non compliant with no address
	description	A restriction on the RETRIEVE of ex:address, violated by T
o address rule 0	label pattern	no-address ex:address restriction P var V W triple Pattern T
		T includes address V
	alt	rule no address default
	assert	Q non compliant with no address
	description	A restriction on the USE of ex:address, violated by T
	label	no-address ex:address restriction
o address rule 1		W triple Pattern
	pattern	T includes V
	h	U includes V @@:_:n153
	assert	
o address default	description label	No restricted attributes found. no-address default rule
	type	Belief Rule

Figure 2-3: Tabulator Representation of the Sample "No Address" Restriction Policy

http://dig.csail.mit.edu/2009/policy-assurance/generator/make_ restriction_policy.py?policyName=no-address&textDescription=Users+may +not+find+the+home+address+of+members+of+the+database.&namespaces=%40 prefix+ex%3A+%3Chttp%3A%2F%2Fexample.com%2F%23%3E.&Attribute1=ex%3A address&Var1=both&ur1=whatever

Figure 2-4: URI for the demo "no address" policy.



Figure 2-5: Converting a SPARQL Query to N3

The administrator visits the translation page to perform the translation:

http://dig.csail.mit.edu/2009/policy-assurance/sparql2n3.py

The administrator enters the query into the translator and gets the result shown in figure 2-6. The translated query appears in a Web browser in figure 2-5.

The administrator clicks the "View in Tabulator" link, to see what the translation looks like in Tabulator. The result is in figure 2-7.

The translation has a unique URI, which is the URI of the Tabulator page. The administrator saves the URI, as in figure 2-8, for checking the query for compliance.

```
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
:Query19369095151250692756 a s:SPARQLQuery;
s:clause [
   s:triplePattern { :s <http://example.com/#ssn> :ssn };
   s:triplePattern { :s <http://example.com/#age> :age };
   s:triplePattern { :s <http://example.com/#name> :name };
   s:triplePattern { :age s:booleanGT "18 "};
];
   s:retrieve [
      s:var :age;
      s:var :name;
      s:var :ssn;
].
```

Figure 2-6: Compliant SPARQL query, converted to N3.

🕹 http://dig.csail.mit.edu/2009/p	oolicy-assurance/pr	int-input.py?input=	@prefix+s:+ <htt< th=""><th>p://dig 💶 🗖 👂</th></htt<>	p://dig 💶 🗖 👂
<u>File E</u> dit <u>V</u> iew Hi <u>s</u> tory <u>B</u> ookmarks	<u>T</u> ools <u>H</u> elp			
🔇 🖻 - C 🗙 🏠 🗋	http://dig.csail.mit.edu	ı/2009/policy-assurance/p	rint-input.pyi 🏠 🔹	Google 🔎
SPARQL to N3 Conversion	🖾 📄 http://di	g.csail.mit.edu/2009/p	ol🛛 +	
*http://dig.csail.mit.edu/2009/policy-assur :Query19369095151250692756+a+s:SPAR ++s:triplePattern++{+:se+s:thtp://example.co ++s:triplePattern++{+:age+s:booleanGT+*1 ++++++s:var+:ssn;].	ance/print-input.py?input QLQuery; s:clause+[++s m/#age>+:age+}; ++s:trij &*'};];+ +++s:retrieve+[=@prefix+s:+ <http: dig.t:<br="">:triplePattern++{+:s+<http: e<br="">plePattern++{+:s+<http: e<br="">++++++s:var+:age; +++++</http:></http:></http:>	sail.mit.edu/2009/IAF :://example.com/#ss xample.com/#name ++s:var+:name; ++++	RPA-PIR/sparql#>+. n>+:ssn+}; >+:name+}; +++s:var+:s;
		age boolean GT	18	
		s age	age	
clause	triple Pattern	s name	name	
Query19389095151250892758		s ssn	ssn	
retrieve	var	age name s		
type	Abstract class of S	PARQL queries.		
type Type CRDFDocument Mentions CAbstract class of SPARQL queries Find All	Abstract class of S	s ssn PARQL queries.		
Done				

Figure 2-7: Tabulator Representation of a Sample SPARQL Query

http://dig.csail.mit.edu/2009/policy-assurance/print-input.py?input= %40prefix+s%3A+%3Chttp%3A%2F%2Fdig.csail.mit.edu%2F2009%2FIARPA-PIR %2Fsparq1%23%3E+.%0A%0A%3AQuery19369095151250692756+a+s%3ASPARQLQuery %3B%0A%0As%3Aclause+[%0A++s%3AtriplePattern++{+%3As+%3Chttp%3A%2F %2Fexample.com%2F%23ssn%3E+%3Assn+}%3B%0A++s%3AtriplePattern++{+ %3As+%3Chttp%3A%2F%2Fexample.com%2F%23age%3E+%3Aage+}%3B%0A++s%3A triplePattern++{+%3As+%3Chttp%3A%2F%2Fexample.com%2F%23name%3E+%3A name+}%3B%0A++s%3AtriplePattern++{+%3Aage+s%3AbooleanGT+%2218+%22}%3B %0A%0A]%3B+%0A+++s%3Aretrieve+[%0A+++++s%3Avar+%3Aage%3B%0A+++ +++s%3Avar+%3Aname%3B%0A+++++s%3Avar+%3As%3B%0A+++++s%3Avar+%3A ssn%3B%0A].%0A%0A

Figure 2-8: URI for the translation of the compliant demo query.

To check the query for compliance against the policy, the administrator goes to the policy execution page:

```
http://dig.csail.mit.edu/2009/policy-assurance/run-policy.py
```

The administrator pastes in the URIs of the query and the policy into the correct text boxes, and clicks "Execute". A "View in Tabulator" link appears, as in figure 2-9, which the administrator clicks. The Tabulator page appears, and the administrator views the output, as in figure 2-10. The administrator can use the justification pane to get more information about this decision, as in figure 2-11.

2.4.3 Checking an Incompliant Query

The administrator now checks an incompliant query:

}

This translation has the URI listed in figure 2-14.

Following the same steps as before, the administrator can see that this query is non compliant, as in figure 2-12 and 2-13.



Figure 2-9: Policy Execute Web Page

bttp://mr-burns.w3.org/cgi-bin/server_cg	i.py?logFile⊧	http://dig.csail.mit.	edu/2009/policy-ass	urance/print-input.p	y?inp 💶 🗖 🔀
Eile Edit View History Bookmarks Tools Hel	p				
🕜 🖂 🗸 🕜 🗋 http://mr-bu	rns.w3.org/cgi-	bin/server_cgi.py?logFile=	http%3A//dig.csail.mit.edu	ı/2009/pol ☆ 🔹 🛂 •	P
Policy Execute	http://mr-bu	rns.w3.org/cgi-bin/ 🖡	3 +		-
* http://mr-burns.w3.org/cgi-bin/server_cgi.py?log%2F%23ssn%3E+%3Assn+}%3B%0A++s%3Atriplei%3B%0A++++s%3Avar+%3Aname%3B%0A++++ %2F%23%3E.&Attribute1=ex%3Aaddress&Var1=b	gFile=http://di Pattern++{+% ++s%3Avar+ ooth&url=wha	g.csail.mit.edu/2009/po "3As+%3Chttp%3A%2F% %3As%3B%0A+++++s (tever	licy-assurance/print-in 62Fexample.com%2F% %3Avar+%3Assn%3B%	put.py?input=%40prefix 23age%3E+%3Aage+}% 0A].%0A%0A&rulesFile=	(+s%3A+%3Chttp%3A% 3B%0A++s%3AtripleP http://dig.csail.mit.edu
no address check		justification	premise		
Query19369095151250692756		compliant with	no address		
g0		closed world assumption	base rules http://dig.csail.mit.edu/20 ++s:triplePattern++{+:age http://dig.csail.mit.edu/T/ organise	109/policy-assurance/print-in +s:booleanGT+"18+"];];+ + AMI/2007/amord/base-assum	put.py?input=@prefix+s:+- ++s:retrieve+[++++++s:var iptions.ttl
		description	No restricted attributes for	und.	
					instance Of
Query19368095151250692756 compliant with	nc address	justification	anteoedent expr	sub expr	justification
			rule name	type no address default	And justification
dause retrieve Query 19389095151250692756 type	g1 g2 Abstract dass of SPARQL queries.	justification	premise		
Type CDFDocument Mentions CAnd justification					
Find All					
«					>
Done					

Figure 2-10: Tabulator Compliance Summary for Sample Query and Sample Policy

e <u>E</u> dit <u>V</u> iew Hi <u>s</u> tory <u>E</u>	ookmarks <u>T</u> ools	Help								
<>> - C × -	http://r	nr-burns.w3.org/cgi-bin/server_cgi.py?logF	le=http%3/	A//dig.csa	ail.mit.ed	lu/200	09/c ₹	3 •	∛ -	
Policy Execute		http://mr-burns.w3.org/cgi-bin/.	🛛 🔶							
nttp://mr-burns.w3.org/cgi \$A%2F%2Fdig.csail.mit.edu A\$%3AClause+[%0A++s% %3As+%3Chttp%3A%2F%2 \$F%23name%3E+%3Aname 0A+++++s%3Avar+%3Aa A%0A&rulesFile=http://dig ktDescription=Users+may	bin/server_cgi.p %2F2009%2FIAR 3AtriplePattern+ Fexample.com% e+}%3B%0A++s% ge%3B%0A++++ i.csail.mit.edu/20 +not+find+the+h	y?logFile=http://dig.csail.mit.edu/2009 PA-PIR%2Fsparql%23%3E+,%0A%0A% +{+%3As+%3Chttp%3A%2F%2Fexamp 2F%23age%3E+%3Aage+}%3B%0A++ 3AtriplePattern++{+%3Aage+s%3Abc :s%3Avar+%3Aname%3B%0A++++++ 09/policy-assurance/generator/make ome+address+of+members+of+the+	/policy-as: 3AQuery1 le.com%2l %3Atriple oleanGT+ s%3Avar+ _restrictio database,	surance 9369099 F%23ssr Pattern+ %2218+ %3As% on_polic &names	/print-ir 515125 n%3E+% ++{+%3. %22}%3 3B%0A+ y.py?pc spaces	nput. 0692 3As: As+9 8B%0 ++++ blicy1 =%40	py?inj 2756+ sn+}% 63Cht A%0A ++s% Name)prefin	put=%4 a+s%3 3B%0, tp%3A]%3B+ 3Avar4 =no-ad <+ex%	40prefix ASPAR A++s%: %2F%2F %0A++ %3Ass Idress& 3A+%30	x+s%3A+%3Ch QLQuery%3B% 3AtriplePatterr Fexample.com +s%3Aretrieve in%3B%0A]. & Chttp%3A%2F
Fexample.com%2F%23%3	E.&Attribute1=ex	x%3Aaddress&Var1=both&url=whate	ver 底	*	2	A	RBF XHL		?	F
luery 19369095151250692756	is compliant with no	address 💌								
Querv19369095151250692756	3 is compliant with (no address								
More Information Start Ov	er									
No restricted attributes t	ound.									
remises:										
entions • • And justification										

Figure 2-11: Tabulator Compliance Justification UI for Sample Query and Sample Policy

🕙 http://mr-burns.w3.org	g/cgi-bin/server_cgi	.py?logFile=http://dig.cs	ail.mit.edu/2009/policy-	assurance/print-inp	ut.py?input 💶 🗖 🔀
<u>Eile Edit View History B</u>	ookmarks <u>T</u> ools <u>H</u> elp)			
C	🏠 📋 http://mr-bur	ns.w3.org/cgi-bin/server_cgi.p	/?logFile=http%3A//dig.csail.mi	t.edu/2009/polic 🏠 🔹	3 • <i>P</i>
Policy Execute					
http://mr-burns.w3.org/cgi- %3B%0A%0As%3Aclause+[% /2009/policy-assurance/gene %3E %Attribute1_cu%3Acdd	-bin/server_cgi.py?log 60A++s%3AtriplePatte erator/make_restrictio	File=http://dig.csail.mit.edu rn++{+%3As+%3Chttp%3A% on_policy.py?policyName=r	2009/policy-assurance/prii 2F%2Fexample.com%2F%2 io-address&textDescriptio	nt-input.py?input=%40 3address%3E+%3Aa+ n=Users+may+not+fin	prefix+s%3A+%3Chttp%3A% }%3B%0A%0A]%3B+%0A+++ id+the+home+address+of+r
%3E.&Attribute1=ex%3Aadd	ress&var1=both&url=	Whatever (
no address check Query4806263041250694128			justification non compliant with	premise no address	
			description	A restriction on the RE	TRIEVE of ex:address, violated b
			description	s address	а
Query4606263041250694128	non compliant with	no address	justification	anteoedent expr	sub expr
					type
·		C		rule name	no address rule 0
Т	includes	s address	а		
Query4606263041250694128	clause retrieve	g0 g1			
	type	Abstract class of SPARQL que	ries.	premise	
80	triple Pattern	s address	а		
g1	var	3			
Type CRDFDocument					
<	101				>
Done		10			

Figure 2-12: Tabulator Non-Compliance Summary for Sample Query and Sample Policy

🕑 http://mr-burns.w3.org/cgi-bin/server_cgi.py?logFile=http://dig.csail.mit.edu/2009/policy-assurance/print-input.py?input 💶 🗖 🔀
<u>File Edit Vi</u> ew History Bookmarks Iools <u>H</u> elp
C X 🟠 🗋 http://mr-burns.w3.org/cgi-bin/server_cgi.py?logFile=http%3A//dig.csail.mit.edu/2009/polic 🏠 🔹 🔀 i 🔎
Policy Execute 🔯 http://mr-burns.w3.org/cgi-bin/ 🔯 🔄 🕶
* http://mr-burns.w3.org/cgi-bin/server_cgi.py?logFile=http://dig.csail.mit.edu/2009/policy-assurance/print-input.py?input=%40prefix+s%3A+%3Chttp%3A%2F%2Fdig.csail.mit.edu/%2F2009%2FIARPA-PIR%2Fsparql%23%3E*0A%0A%3AQuery4606263041250694128+a+s%3A5PARQLQuery%3B%0A%0A%3AQuery4606263041250694128+a+s%3A5PARQLQuery%3B%0A%0A%3ASuery4606263041250694128+a+s%3A5PARQLQuery%3B%0A%0A%3ASuery4606263041250694128+a+s%3Aretrieve1%0A+++++s%3Avar*%3A5%3B%0A++++++s%3Avar*%3A5%3B%0A,%0A%3ASuery4606263041250694128+a+s%3Aa+}%3B%0A
Query4806263041250694128 is non compliant with no address
More Information Start Over
A restriction on the RETRIEVE of ex:address, violated by { <http: 2009="" dig.csail.mit.edu="" policy-assurance="" print-<br="">input_py?input=%40prefix+s%3A+%3Chttp%3A%2F%2Fdig.csail.mit.edu%2F2009%2FIARPA-PIR%2Fsparql%23%3E+. %0A%0A%3AQuery4606263041250694128+a+s%3ASPARQLQuery%3B%0A%0As%3Aclause+ [%0A++s%3ArtiplePattern++{+%3As+%3Chttp%3A%2F%2Fexample.com%2F%23address%3E++%3Aa+}%3B%0A %6A]%3B+%0A+++s%3Aretrieve+[%0A+++++s%3Avar+%3Aa%3B%0A++++++s%3Avar+%3As%3B%0A].%0A%0A#s> <http: #address="" example.com=""><http: 2009="" dig.csail.mit.edu="" policy-assurance="" print-<br="">input_p?input=%40prefix+s%3A+%3Chttp%3A%2F%2Fexample.com%2F%2FARPA-PIR%2Fsparql%23%3E+. %0A%0A%3AQuery4606263041250694128+a+s%3ASPARQLQuery%3B%0A%0As%3Aclause+ [%0A++s%3AtriplePattern++{+%3As+%3Chttp%3A%2F%2Fexample.com%2F%23address%3E+3Aa+}%3B%0A %0A]%3B+%0A+++s%3Aretrieve+[%0A+++++s%3Avar+%3Aa%3B%0A+++++s%3Avar+%3As%3B%0A].%0A%0A#a>.} Make sure this is a query.</http:></http:></http:>
Premises:
No more information available from the reasoner!
Type CRDFDocument
Find All
http://mr-burns.w3.org/cgi-bin/server_cgi.py?logFile=http%3A//dig.csail.mit.edu/2009/policy-assurance/print-input.py?input=%2540prefix%2Bs%253A%2B%253Chttp%253A%

Figure 2-13: Tabulator Non-Compliance Justification UI for Sample Query and Sample Policy

http://dig.csail.mit.edu/2009/policy-assurance/print-input.py?input=%
40prefix+s%3A+%3Chttp%3A%2F%2Fdig.csail.mit.edu%2F2009%2FIARPA-PIR%2F
sparql%23%3E+.%0A%0A%3AQuery4606263041250694128+a+s%3ASPARQLQuery%3B
%0A%0As%3Aclause+[%0A++s%3AtriplePattern++{+%3As+%3Chttp%3A%2F%2F
example.com%2F%23address%3E+%3Aa+}%3B%0A%0A]%3B+%0A+++s%3Aretrieve+
[%0A+++++s%3Avar+%3Aa%3B%0A+++++s%3Avar+%3As%3B%0A].%0A%0A

Figure 2-14: URI for the translation of the non-compliant demo query.
2.4.4 Demo Notes

This demonstration will work on any Firefox Web browser with a recent version of the Tabulator plugin installed. Without the Tabulator plugin, the demo will work in any Web browser. However, withou the plugin, a browser will either display the textual output of the reasoner, or prompt the user to save the output to a file on disk.

2.5 Summary

This chapter presented an overview of the system implemented in this thesis. It described a sample scenario, and defined three perspectives to the system. It provided a Web based demonstration of the three major components of the system: the SPARQL to N3 query converter, the automated policy generator, and the user-friendly reasoner output. The following chapters offer more technical information about the design and implementation of this system.

Chapter 3

System Detail

In this chapter, we discuss the implementation details of the policy assurance system. Each section details a particular component of the system, discussing the design assumptions, choices, and tradeoffs that were necessary. This chapter serves as full and complete documentation of this project for future users and developers of the system.

This section benefits from a working knowledge of RDF, N3, AIR, SPARQL, Python, and C++. The code to support this discussion is included in the appendix.

3.1 SPARQL Query Translation

The policies that we defined in AIR in the previous section require SPARQL queries to be in a particular N3 syntax in order to correctly perform reasoning. The design of the AIR policy and the SPARQL query translation go hand in hand. The first approach at query reasoning used a SPARQL translation that maintained the semantics of the SPARQL query, in fulfillment of the Phase we milestone for this project. In time, we moved to a translation that removes many of the semantics of SPARQL queries. We discuss the details of the new translation herein.



Figure 3-1: A screenshot of the SPARQL to N3 translator Web page.

- :SPARQLQuery. Refers to a query converted to N3.
 - :source. A data source for the query. In SPARQL, this can be FROM or FROM NAMED. A :SPARQLQuery may have many of these.
 - :retrieve. A list of variables to which the :SPARQLQuery refers. A :SPARQLQuery has exactly one of these.
 - * :var. A member of the :retrieve which describes a particular variable.
 - :clause. The contents of the :SPARQLQuery. This corresponds to the WHERE clause in SPARQL. A :SPARQLQuery has exactly one of these.
 - * :triplePattern. A particular line or binding in a :SPARQLQuery.

Table 3.1: Query conversion ontology.

3.1.1 SPARQL to N3 Web Page

The SPARQL to N3 Web page serves as a front-end to the conversion process discussed in the previous section. The user inputs a well-formed SPARQL query into the form on the page. Upon clicking submit, the page's backend script will pass the query to the translator, a command-line based utility. The translator then returns an N3 translation of the user's SPARQL query to the Web page.

A user may use the SPARQL to N3 translator directly to help debug the policy process, or to gain understanding of the query process. It is important to note that the N3 translation does not preserve all SPARQL semantics, nor does it preserve order in some cases. In the majority of use cases, the user will interact indirectly with the translator.

3.1.2 Query Conversion Ontology

We define an ontology for N3 conversion of generic queries in table 3.1. All of these names are defined in the query conversion namespace, which is:

http://dig.csail.mit.edu/2009/policy-assurance/sparql.n3#

The ontology is intentionally small, short, and simple. Whereas the first iteration of query translation focused to have a bidirectional, complete N3 translation of a

Abstract Sparql Ontology



Figure 3-2: Ontology diagram of the SPARQL translation, courtesy of Yotam Aron.

SPARQL query, the goal of this ontology is to facilitate the conversion of queries in additional languages, and to contain the minimum amount of information necessary so as to continue to be useful in making assertions according to our current policies. (An important future work of this project, and perhaps the largest hurdle to widespread adoption, is the support of SQL queries.)

The next subsections will demonstrate exactly how we convert a query.

3.1.3 swobjects: Parsing and Serializing

The current implementation of the SPARQL to N3 translator is an adaptation of Eric Prud'Hommeaux's swobjects code [11]. swobjects, per its homepage, is a suite of tools written in C++ for performing operations upon Semantic Web objects. swobjects is an important part of the SPASQL project, a project whose goal is to facilitate data integration from multiple databases [12].

This project modifies the serialization engine of swobjects. The swobjects code, in one mode of operation, will accept a SPARQL query as input, and print the same SPARQL query as output. The swobjects code performs parsing and lexing of the input, and creates a parse tree that it expresses. By overloading the classes of the SPARQLSerializer code in swobjects, we were able to modify the output of a SPARQL query. By carefully defining the output and performing a small amount of additional processing, we are able to perform conversion.

The C++ code that overloads the SPARQLSerializer is located in the DIG repository at

http://dig.csail.mit.edu/2009/policy-assurance/sparql2n3.cpp

3.1.4 SPARQL Language Translation

In this section, we discuss the details of translating a SPARQL query to N3. The discussion closely follows the W3C Recommendation for SPARQL [4], and uses some queries from the DAWG test case [13]. We discuss which features we support, and how we support them. We explain which features of the language we do not support,

and why we chose the approach that we did in a few situations. The translation is incomplete; in particular, the GRAPH feature does not work, nested queries do not work, and the FILTER keyword is only partially supported. Furthermore, whitespace generation needs improvement. Nevertheless, this translation is suitable for the majority of simple SPARQL queries.

As of this writing, the query translator is live. To use it, please visit:

http://dig.csail.mit.edu/2009/policy-assurance/sparql2n3.py

Namespaces and URIs: BASE and PREFIX

A SPARQL query may specify its own namespace using the @prefix keyword, or the @base keyword:

@prefix dc: <http://purl.org/dc/elements/1.1/> .

A namespace is shorthand, allowing truncation of URIs. For lack of ambiguity, we kept the swobjects feature of expanding or flattening all URIs. The output of the translation only specifies one name space: the namespace of the translation ontology.

Query Identification

Every query that goes through the translator is assigned a unique name. In the current implementation, we cannot be assured of the uniqueness of the name, but as it is assigned by a random number plus the current time stamp (seconds since UNIX epoch), the probability of 1 in 2^{32} of a collision in one second is acceptably low. A one-way hash of a query does not work, as this causes successive translations of the same query to have the same name. It is conceivable that a user might run the same query, or a series of very similar queries, an arbitrary number of times. Our policies depend on uniquely named policies to correctly determine which policies are in compliance or violation of a policy.

SELECT

A SPARQL query with a SELECT clause "returns all, or a subset of, the variables bound in a query pattern match." Thus, a query with a SELECT clause has a variable part and a triple pattern part.

If we input the query:

The translator will generate the N3 output:

```
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
```

```
:Query9743408551250696473 a s:SPARQLQuery;
```

```
s:clause [
  s:triplePattern { :s <http://example.com/#ssn> :n };
  s:triplePattern { :s <http://xmlns.com/foaf/0.1/age> :a };
  s:triplePattern { :s <http://xmlns.com/foaf/0.1/openid> :id };
];
  s:retrieve [
    s:var :n;
    s:var :s;
].
```

SELECT *

The syntax SELECT * is an abbreviation that selects all of the variables in a query [4]. To implement this, the translator adopts a special behavior when it sees SELECT *. It will maintain a set of all variables mentioned in the body of the query. At the end of the clause, after processing is complete, the translator will print a list of every variable seen. In effect, this implements the correct handling of SELECT *, in line with the "flattening" philosophy of the translator.

Given this example query, with a SELECT *,

The translator will generate the N3 output:

```
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
:Query14924979641250696494 a s:SPARQLQuery;
s:clause [
   s:triplePattern { :s <http://example.com/#ssn> :n };
   s:triplePattern { :s <http://xmlns.com/foaf/0.1/age> :a };
   s:triplePattern { :s <http://xmlns.com/foaf/0.1/openid> :id };
];
s:retrieve [
   s:var :a;
   s:var :id;
   s:var :n;
   s:var :s;
].
```

CONSTRUCT

A CONSTRUCT query is very similar to a SELECT, except in that it creates a new RDF graph with its output. In effect, it is a union of all of the solution sets. Thus, the useful output of a CONSTRUCT query is the same as a similar SELECT query, in that it will only output explicit and not blank variables. The translator handles variables in a CONSTRUCT similarly to a SELECT *: it will output every non-blank variable mentioned in the CONSTRUCT part of the clause.

Given this example query, with a CONSTRUCT,

CONSTRUCT { ?s <p1> <o> . ?s <p2> ?o } WHERE {?s ?p ?o}

The translator will generate the N3 output:

```
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
:Query13731259441250696648 a s:SPARQLQuery;
    s:clause [
    s:triplePattern { :s :p :o };
];
    s:retrieve [
        s:var :o;
        s:var :s;
].
```

Note that the variable list is always placed *after* the clause, due to the current implementation of the translator. The reasoner will accept the variable list before or after the clause without loss of functionality.

ASK

In an ASK query, the user may "test whether or not a query pattern has a solution. No information is returned about the possible query solutions, just whether or not a solution exists." [4] Thus, no variables are bound or output in an ASK clause. Of course, an ASK query may still explicitly refer to a particular field or data type which we may wish to regulate, so it is important to capture that pattern in the output.

Given this example query, with an ASK,

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK { ?x foaf:name "Alice" }
```

The translator will generate the N3 output:

@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .

:Query5293946291250696665 a s:SPARQLQuery;

```
s:clause [
   s:triplePattern { :x <http://xmlns.com/foaf/0.1/name> "Alice" };
].
```

This is sensible, because even though we don't directly output any data from the database, we are asking about "Alice" and learning something about a foaf:name.

DESCRIBE

The SPARQL DESCRIBE construct is implementation dependent. A particular SPARQL endpoint may implement DESCRIBE as it sees fit. From a translation and policy standpoint, it is possible that DESCRIBE query will return information about the variables and triple patterns it describes. Thus, we treat DESCRIBE as we would a SELECT.

Given this example query, with a DESCRIBE,

```
DESCRIBE <u> ?u WHERE { <x> <q> ?u . }
```

The translator will generate the N3 output:

@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .

```
:Query2577564471250696683 a s:SPARQLQuery;
<u>s:clause [
    s:triplePattern { <x> <q> :u };
];
    s:retrieve [
        s:var :u;
].
```

Query Modifiers: ORDER BY, LIMIT, OFFSET, DISTINCT, REDUCED

SPARQL supports a number of query modifiers that alter the output. ORDER BY serves to sort the output by its argument. LIMIT reduces the number of patterns that a query may bind. OFFSET works with ORDER BY and LIMIT to select a subset of the full output. DISTINCT and REDUCED change how a SPARQL query will handle duplicates.

These query modifiers form important parts of SPARQL's semantics. However, from a policy standpoint, their results are highly data dependent. Since the translation must be as pessimistic as possible, it simply drops these keywords from the output.

OPTIONAL

The OPTIONAL keyword serves to make a query pattern conditional. Normally, individual query triple patterns are mandatory, and all of them must be satisfied if they chain together. An OPTIONAL pattern is not mandatory. It will return results if they are present in the store, but will not fail if there is no data to support it.

Since our translator must take a pessimistic stance, we simply drop the OP-TIONAL keyword from the translation. Our policies must be able to detect if there is *any possibility* that a triple pattern will result in a binding. It is unimportant whether or not the pattern is mandatory if we assume that it will have a binding.

UNION

The UNION keyword allows the user to specify alternatives to a query pattern binding. This allows multiple results to be concatenated. The translator simply drops the UNION keyword, since policies do not need to use its semantics.

Given this example query, with a UNION,

Operator	Name	Translation
-	arithmeticNegation	:arithmeticNegation
/	arithmeticInverse	: arithmeticInverse
+	$\operatorname{arithmeticSum}$: arithmetic Sum
*	arithmeticProduct	:arithmeticProduct
&	booleanConjunction	:booleanAND
	booleanDisjunction	:booleanOR
!	booleanNegation	:booleanNOT
=	boolean EQ	:booleanEQ
!=	booleanNE	:booleanNE
<	booleanLT	:booleanLT
>	booleanGT	:booleanGT
<=	booleanLE	:booleanLE
>=	booleanGE	:booleanGE

Table 3.2: Conversion of Boolean operators. The "name" refers to the title of the operator in the translator code. The "translation" is what the translator outputs, in the query translation namespace.

The translator will generate the N3 output:

```
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
```

```
:Query15319361881250696825 a s:SPARQLQuery;
```

```
s:clause [
    s:triplePattern { :book <http://example/1.0/title> :title };
    s:triplePattern { :book <http://example/1.1/title> :title };
```

```
];
s:retrieve [
s:var :title;
].
```

Boolean Functions

SPARQL FILTERs support a number of boolean operators. The translation changes those operators into terms defined in its namespace. This allows the writing of policies that can detect these operators. A future work is to declare these as equal to other operators in other ontologies. The current translation is in table 3.2.

Built-in Functions: STR, LANG, LANGMATCHES, DATATYPE, BOUND, sameTERM, isURI, isIRI, isLITERAL, REGEX, true, false

SPARQL supports a number of built in functions for query processing. Their primary use is to serve as arguments to the FILTER keyword. Our translation does not support these built in functions at present.

FILTER

The FILTER keyword in SPARQL allows the user to restrict a query's output based on a condition. The condition may be a triple pattern, which we fully support and exemplify below. Alternately, the condition may include a built-in function, which we do not support; the most popular form if this is FILTER REGEX. In terms of writing policies, a FILTER counts as a "use" of a policy. In the case of a triple pattern, our translator drops the FILTER keyword, as its semantics are unimportant to the translator, and includes the triple pattern from the FILTER.

Given this example query, with a FILTER,

```
SELECT * WHERE { FILTER (?o>5) . ?s ?p ?o }
```

The translator will generate the N3 output:

```
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
```

:Query10909264011250696848 a s:SPARQLQuery;

```
s:clause [
  s:triplePattern { :s :p :o };
  s:triplePattern { :o s:booleanGT "5 "};
];
  s:retrieve [
    s:var :o;
    s:var :p;
    s:var :s;
].
```

GRAPH

The GRAPH keyword restricts a particular pattern to only apply to unnamed (using the FROM keyword) or named (using the FROM NAMED keyword) graphs. If used in conjunction with a named graph, the result will note which graph contained the matching result. At present, the translation does not support the GRAPH keyword. It requires some further consideration, as it is difficult to express and complicates a query.

One possibility is to simply drop it; this would easily support all of the semantics of an unnamed graph query, and allow us to write policies that simply check which graphs we include with FROM or FROM NAMED. Another, more sophisticated, approach would be to generate several equivalent queries, in effect flattening the GRAPH keyword. With multiple uses of GRAPH in a query, this could easily grow exponentially.

3.1.5 Lost in Translation

As mentioned earlier, our original translation maintained more SPARQL semantics. This approach is deprecated in favor of the approach described here, but for completeness, we include a SPARQL query with translations under the old and new translations.

Given this SPARQL query as input,

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dig.csail.mit.edu/2009/IARPA-PIR/generic#>
SELECT ?s ?b ?c WHERE {
    ?s type:b ?b.
    ?s type:c ?c.
}
```

the first version of the translation would have yielded this output,

@prefix type: <http://dig.csail.mit.edu/2009/IARPA-PIR/generic#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

```
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
@prefix : <http://dig.csail.mit.edu/2009/IARPA-PIR/query1#> .
:Query-1 a s:Select;
    s:cardinality :ALL;
    s:POSList [
        s:variable :S;
        s:variable :B;
        s:variable :C;
    ];
    s:WhereClause :WHERE.
:WHERE a s:DefaultGraphPattern;
        s:TriplePattern { :S type:B :B };
        s:TriplePattern { :S type:C :C }.
```

#ends

whereas the current version of the translator yields this output.

@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .

:Query6088797851250696900 a s:SPARQLQuery;

```
s:clause [
s:triplePattern { :s <http://dig.csail.mit.edu/2009/
IARPA-PIR/generic#b> :b };
s:triplePattern { :s <http://dig.csail.mit.edu/2009/
IARPA-PIR/generic#c> :c };
];
s:retrieve [
s:var :b;
s:var :c;
s:var :s;
].
```

The first version closely preserved most, if not all, of the SPARQL semantics. We found this approach to be cumbersome, as it required writing brittle policies that closely followed SPARQL query semantics. The old approach did not scale well, and did not lend itself to automated policy generation.

3.1.6 Translator Summary

The translator presented herein is novel, in that we know of no prior attempt to express a SPARQL query in N3 for any reason. The implementation of the translator offers insight as to exactly what parts of an arbitrary query give it structure and permit reasoning. We found that most of the SPARQL semantics simply led to more complicated policies, and were redundant.

3.2 AIR Policy Generation

3.2.1 Templates for Policy Generation

The goal of this project is to make policy assurance accessible to a user with an intermediate level of technical understanding, without a thorough grounding in Semantic Web technologies. To this end, this project offers a number of templates that allow a user to directly create policies according to several fixed design patterns. With a list of attributes as an input, the policy generator will create a valid AIR policy to facilitate reasoning.

All of these templates work on the level of single attributes. Furthermore, the policies that the templates generate rely on matching very particular triple patterns. Though users can represent many policies with a combination of one or more templates, there are policies that the templates cannot describe. These patterns cover some basic cases, and provide a starting point for more advanced policies. A description of some of the common design patterns follows. An advanced, AIR-savvy user is still able to create their own policies.

3.2.2 Supported Policy Types

At this time, this project defines five separate kinds of policies. These policies are designed to be templates, to help an administrator create or implement more sophisticated policies. These policies do not necessarily exhaust the space of policies that this system can represent, but serve as a basis for many policies. Each policy works by matching a particular variable type or attribute, and taking action when it finds that type. The five policies, in no particular order, perform as described on variable types that a user enters.

- **Restriction**. Blocks access to a variable type.
- Inclusion. Lumps variable types together, so a user must access either all of them or none of them.
- Exclusion. Allows a user access to all but one of the specified variable types.
- Chaining. Accepts a list of variable types. If the user mentions the first variable type, chaining takes action if it sees any of the following variable types.
- **Default Deny**. Restricts access to the specified variable types.

The following sections describe the implementation details of each of these policies.

To USE and to RETRIEVE

All of the policy templates in this section implement patterns which detect two common usage patterns, USE and RETRIEVE. To USE an attribute is to pull it from a database and make decisions based on its value. To RETRIEVE an attribute is to pull it from a database and display it directly to the user. It is possible to USE an attribute without a RETRIEVE, and vice versa.

When a user submits a query to USE a variable, the policy detects this by searching for a particular triple pattern:

```
:W s:triplePattern :T;
:T log:includes { [] example:age :V };
:W s:triplePattern :U;
:U log:includes { :V [] [] }.
```

The policy is looking for a pair of triple patterns to detect usage. In the first triple pattern, a variable is the object of a triple pattern, with an attribute as the predicate.

In the second triple pattern, the variable that was the object of the first pattern is now the subject. This ascertains that a user bound a variable to a certain attribute, and is trying to use that same variable for further processing. As an example, the following query would match this pattern.

```
# Sample SPARQL query that queries for SSN, age, OpenID.
PREFIX example: <http://example.com/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
    ?s example:ssn ?n. ?s foaf:age ?a. ?s foaf:openid ?id.
    FILTER (?a > 18) }
```

This query, translated into N3 with the aforementioned converter, appears as follows.

@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .

```
:Query-1644923176 a s:SPARQLQuery;
    s:retrieve [
    ];
s:clause [
    s:triplePattern { :s <http://example.com/#ssn> :n };
    s:triplePattern { :s <http://xmlns.com/foaf/0.1/age> :a };
    s:triplePattern { :s <http://xmlns.com/foaf/0.1/openid> :id };
    s:triplePattern { :a s:booleanGT "18 "};
```

```
];
```

```
s:retrieve [
    s:var :a;
    s:var :id;
    s:var :n;
```

s:var :s;

].

In particular, the USE policy would find two triple patterns of interest, indicating that the user's query is trying to USE something with an attribute of foaf:age:

```
s:triplePattern { :s <http://xmlns.com/foaf/0.1/age> :a };
s:triplePattern { :a s:booleanGT "18 "};
```

A user may RETRIEVE an attribute independently of trying to USE it. When trying to RETRIEVE, a user will submit a query that binds an output variable to a particular attribute. In other words, the user submits a SPARQL query for verification, with a {subject, predicate, object} triple in which the predicate is a type (specifically, a URI, like example:dob), and the object is a variable, like :V. The reasoner uses this information to decide, ":V is a variable that will match things of type example:dob." The policy looks at variables in the context of attributes in the database, instead of the actual, bound values within the database itself, since our policies never look inside the database.

The following code demonstrates an AIR pattern that will catch a user trying to RETRIEVE an attribute with a type example:dob.

```
:P s:var :V;
:W s:triplePattern :T;
:T log:includes { [] example:dob :V }
```

To RETRIEVE an attribute, a user must explicitly SELECT a variable (or SELECT *), and bind it in a triple pattern as the object, with the attribute as the subject. In the aforementioned query, a user's attempt to RETRIEVE an attribute of type example:ssn is evident with these N3 statements.

```
s:retrieve [
    s:var :n; ]
s:clause [
    s:triplePattern { :s <http://example.com/#ssn> :n }; ]
```

The important distinction between **RETRIEVE** and **USE** is that the former allows a user to access data in a data set directly, while the latter does not. Once a user has seen a particular piece of data, the policy assurance framework considers it to be "public". There may be incentives in place for the user to control data retrieved from a data set, but those incentives are external to the policies.

The distinction between USE and RETRIEVE is intended for honest, but curious, users, to help keep them honest. An attacker could easily write a series of USE policies that return data that is equivalent to a RETRIEVE. For example, the author's calculations based on publicly available Social Security Administration data shows that there are 746,347,500 possible Social Security numbers. If a policy prevents a user from directly seeing a Social Security number in a database, the user could simply exhaust the SSN space using USE queries. An administrator can specify policies that limit the number of times a user can access a particular attribute. A true security approach with a malicious user in mind is an important future work, well outside the scope of this thesis.

With these common access patterns defined, it is possible to write template policies that can flag these accesses.

The discussion of sample policy templates uses possible fields from a database of personal information. Specifically, it uses example:name for a person's name, example:age for a person's date of birth, example:dob for a person's date of birth, and example:ssn for a person's social security number. In practice, a system of realworld policies would require multiple AIR policies; an important future work would help a user with this task.

The template policies are restriction, inclusion, exclusion, history-aware exclusion, chaining, and default deny. A description of each template policy follows. Samples of all of these policies are available in the appendix. Furthermore, tables which demonstrate how these ideas become policies appear at 3.3 and 3.4.¹

Restriction

A restriction policy disallows a user from performing a USE or RETRIEVE operation (or both) concerning any data with a particular attribute. The policy implicitly implies that a user can see all of the data in a data set, with the exception of the restrictions put in place by a restriction policy. Restriction is most helpful for controlling the way a user interacts with data of a particular attribute.

The sample restriction policy in appendix section B.6 states that a user:

- May not **RETRIEVE** any attribute of type **example:name**.
- May not USE any attribute of type example:age.
- May not **RETRIEVE** any attribute of type **example:dob**.
- May not USE any attribute of type example:ssn.

A user may, for example, USE an attribute of type example:name, since it is a pattern that the policy does not mention.

The policy runner will assert that the user is not in compliance with this policy if it finds any one of those usage patterns in a query. It will return the decision, with an explanation. The policy runner does not evaluate patterns in any particular order. In practice, a restriction policy would be most useful with a default deny policy in place.

Inclusion

An inclusion policy requires that a user tie together certain actions. Inclusion implements an all or none policy that is only available using two or more attributes. An administrator might use this policy to ensure that two fields in a data set are always coupled.

The sample restriction policy in appendix section B.7 states that a user's query is compliant if the query will:

• RETRIEVE any attribute of type example:name, AND

- USE any attribute of type example:age, AND
- RETRIEVE any attribute of type example:dob, AND
- USE any attribute of type example:ssn.

If a query contains any subset of these patterns, but not all of them, the query is incompliant. If the query does not contain any of these patterns, it is compliant.

In practice, an inclusion policy likely has little use. A policy that requires a mandatory filter, such as an age filter, will use chaining and not inclusion.

Exclusion

An exclusion policy is, in effect, the inverse of an inclusion policy. It allows an administrator to tie together a number of patterns. Under an exclusion policy, a user's query (or query history) may contain "all but one" of the patterns in the policy.

The sample exclusion policy in appendix section B.8 states that a user's query is not compliant if the query will:

- 1. RETRIEVE any attribute of type example:name, AND
- 2. USE any attribute of type example:age, AND
- 3. RETRIEVE any attribute of type example:dob, AND
- 4. USE any attribute of type example:ssn.

A user's query may access all but one of these patterns with an AND relation, or none of them at all. As an example, the user can RETRIEVE example:name and RETRIEVE example:dob and USE example:ssn, either in one query or in successive queries if history is enabled. If the user does this, they will not be able to USE example:age. Exhaustively, if we use the notation $\{a \land b \land ...\}$ to express a user trying to access attributes from the above list, the user may NOT do $\{1 \land 2 \land 3 \land 4\}$, but may do any subset of that, including $\{\}$ (nothing). Exclusion is the only basic policy construct for which it makes sense to use a history. The sample policy in appendix section B.9 demonstrates how a policy needs to change in order to support query history. In a break from all other policies, a history-aware policy needs to have an idea of which is the current query, and which are the past queries. This is important, since AIR, a production rule system, does not support any concept or order or counting. A history-aware policy will assert non-compliance if it matches all of the patterns in the policy, regardless of whether those patterns are contained solely in the history, solely in the current query, or in some combination of current and past queries. A full explanation of query history follows in the next section.

Originally, this project sought to include a MAX(m, n) construct. Exclusion implements a MAX(n - 1, n) policy, where a user can see at most n - 1 out of npatterns. Implementing the more general MAX(m, n) for $m \le n$ and n > 0 involves combinatorics and does not scale well. Finding an efficient, scalable implementation of general MAX(m, n) is an important future work.

Chaining

The chaining policy allows an administrator to implement policies in the form IF THEN; to detect the presence of a pattern and determine compliance or non compliance based on the presence of future patterns. A chaining policy can either be default compliant (in which case a pattern match asserts non-compliance), or default non-compliant. The code for the chaining policy generator supports **RETRIEVE** and **USE**, as well as filter patterns.

The sample chaining policy in appendix section B.10, a default non-compliant policy, will try to match the first pattern. If it succeeds, it will make an assertion if it finds any of the successive patterns, and will go to the default if it finds none of them. It walks though the patterns one after the other, as links in a chain. The sample policy states:

• If a user wishes to RETRIEVE any attribute of type example:name, the user must

- USE an attribute of type example:age, OR
- RETRIEVE an attribute of type example:dob, OR
- USE an attribute of type example:ssn, OR
- Include a filter of the form example:age > 18.

More generally, chaining specifies a policy of the form "if A matches, take action if B or C or ... is found." A, B, C, ... can be either a USE or RETRIVE, and "take action" is either "assert compliance" or "assert noncompliance". To match a pattern that is not some form of USE or RETRIEVE, the administrator would need to code this manually; there is no facility in the user interface for the input of a pattern. Furthermore, since the reasoner only looks at the queries to a system, and not their results, making policies that fire if certain data is in the database is not possible. As an example, consider a policy that says, "you may not RETRIEVE any attribute of type example:name if the value of name is John." We can't check to see that a name has value John, so we can't use exclusion here. What we might do instead is to create a policy that requires a user to specify "example:name ! = John" in every query they submit.

Chaining is helpful when an administrator would like to provisionally provide access to a certain attribute. With a chaining policy, the administrator can require a particular filter or other pattern to be present in order for compliance. Since this works at the granularity of a single query, using a history does not make sense for this policy!

Default Deny

The last of the primitives for automatic policy generation, default deny allows an administrator to impose limitations on all queries. Whereas all of the previous policies made no assumption about the size of the world by assuming an open world, default deny imposes strong restrictions by closing the space of possible queries to only match a few particular patterns. This is analogous to the **GRANT** feature of SQL databases, which explicitly enables permissions to a particular field. This policy family is called "default deny" because any pattern that it does not expressly enumerate is, by default, not compliant with the policy.

The sample default deny policy in appendix section B.11 states that a user's query is compliant only if every single triple pattern in its translation matches one of the following patterns, where [1] is a wildcard that can match anything:

- [] example:name []
- [] example:age []
- [] example:dob []
- [] example:ssn []

Implicitly, this policy allows a user to RETRIEVE or USE any of these attributes, to use filters, or to peform any operation that specifically matches one of those four patterns.

In practice, a default deny policy can set the ground rules for access to a particular data set. By explicitly giving permissions to certain triple patterns, the administrator can guarantee closure on the policies. In conjunction with the other template policies, an administrator can offer a level of access between 'all' and 'none'.

3.2.3 Automatic Policy Generation

The table that follows demonstrates how the aforementioned policy templates generate policies based on user input. The purpose of the table is to show how the generation of all policies is similar, and how each policy scales with multiple variables. The author prepared these tables to provide a reference for implementing each policy template.

There are two tables: one for USE policies, and one for RETRIEVE policies. Along the top row of each table are the five policy types. Each table is divided in half: the top half of each table demonstrates what happens if a user enters a single variable type or attribute (say type:A), and the bottom half shows what happens if a user enters two or more variable types or attributes (say type:A, type:B, and possible type:C). The three rows - AIR Rule, Description, Action - explain how a policy gets generated, and what it does. The AIR Rule(s) row shows the snippet of AIR code that forms the important part of the policy. Some policies require several AIR rules to express their functionality, particularly those with multiple variables. The table illustrates AIR code that would appear in separate rules by separating it with a blank line (two newlines). In reality, each policy contains a large amount of boilerplate code, but this snippet really defines the policy. The Description row offerse a terse explanation of the effect of the policy, and the Action row shows what action the reasoner takes when it matches the code in the Rule row. All of these policies depend on a boilerplate that defines :Q as a s:SPARQLQuery, with a s:retrieve part :P and a s:clause part :W.

As an example, consider a policy where an administrator wants to create an Inclusion policy, which specifies that a user must either USE type:A and type:B together in a query, or not at all. The chart demonstrates how the policy generator will create this policy. This is a USE policy, so the template is listed in the USE table. It is an Inclusion policy, which is the third column from the left. This policy mentions multiple variables, so the information that describes it is in the bottom half of the table. In particular, three cells describe the AIR Rule(s), the Description, and the Action of this policy. The AIR Rule(s) row tells us that this policy will have three separate AIR rules that perform pattern matching: one that looks for both type:A and type:B used together, and one each looking for type:A or type:B. The Description row tells us what the policy tries to accomplish, and the Action row describes the action the reasoner takes upon a match between the policy and the query.

Some policies support a "hybrid", whereby a policy can act on a query that tries to USE some attribute(s) and tries to RETRIEVE some other attribute(s). These cases are listed in the Notes row of the USE table. Some policy types do not make sense, so the table marks them as unsupported.

	USE Restriction	USE Inclusion	USE Exclusion	USE Chaining	USE Default Deny
:W s:tripl :T log:inc :W s:tripl :U log:inc	eFattern :1. ludes [] type:SSN :V . ePattern :U. iludes :V [] [] .	Unsupported.	Unsupported.	w stripletatten : 11. :Tl log:includes [] type:A :V1 . :W striplePatten :U1 . :U1 log:includes :V1 [] [] . :W s:triplePattern :T2.	:W stripteFattern :I. :T log:notIncludes [] type:A []
Cannot u	se anything of type:A.	1		:T log:includes [] type:B :V2 . :W s:triplePattern :U2. :U2 log:includes :V2 [] [] . T using type:A, must not use type:B.	May only use type:A.
air:asser with :W	t :Q air:non-compliant- hatever-Policy ;	1		If first rule matches, assert non-compliance if second rule matches. Default is compliance. (Can have negated version as well.)	Assert non-compliance, since you found something other than type:A. No distinction between USE and RETRIEVE.
W sith W sith W sith W sith W sith U log W sith U log U log i U log i U log i U log i T log i I I I I I I I I I I I I I I I I I I	<pre>plePattern T.I. plePattern UI. includes [] type:A :V1 . plePattern :U2. includes [] type:B :V2 . plePattern :U2. includes :V2 [] [] . includes :V2 [] [] . andthes :V2 [] [] . hattern :U2. includes :V2 [] [] . includes :V2 i] [] .</pre>	W sitriplePattern .T1. .T1 log:includes [[type:A :V1.]. .W1 log:includes [[type:B :V2.] .W1 log:includes [[type:B :V2.] .W1 log:includes [[type:B :V2.] .W2 log:includes [[type:B :V2.] .W2 sitriplePattern :U2. .W2 sitriplePattern :U1. .W1 log:includes [[type:A :V1.] .W1 log:includes [[type:B :V2.] .W2 sitriplePattern :U2. .W2 sitriplePattern :U2. .W3 sitriplePattern :U3. .W3 sitriplePattern :U	: W striplePattern :T1. :T1 log:includes [] vpe:A :V1 . .U1 log:includes [] vpe:A :V2 . .U2 log:includes [] vpe:B :V2 . .W striplePattern :U2 . .W striplePattern :U2 . .U2 log:includes :V2 [] [] . .Cannot use type:A and type:B. Generalizes to MAX(N-1,N) Assert non-compliance.	If use: I'l use: I'l use: I'l use: I'l use: I'l use: I'l log: I'l use: I'l the I'l first rule matches, assert non- compliance if any of the follow- pliance. (Can have negated ver- pliance. (Can have negated ver- sion as well.)	 :W striplePattern :T. :T log:notIncludes [] type:A [] :T log:notIncludes [] type:B [] . May only use type:A or type:B. Masert non-compliance, since you found something other than type:A or type:B. No distinction between USE and RETRIEVE.
:Q a s:S s:retriev s:clause	PARQLQuery; .e :P; :W.	We can create hybrid policies, i.e. "if using type:A must re- trieve type:B and vice versa".	We can create hybrid policies, i.e. "may do all but one of: use type:A, retrieve type:B".	Hybrids possible. Can only have one type on the left hand side. Doing an AND is easy, the OR involves combinatorics. We can have mandatory filters us- ing this policy.	Hybrids possible. Can simplify to [] type:X [] for "any access". USE or RETRIEVE are special cases.

Table 3.3: Automatic policy generation: USE.

Notes						Multiple				Single	
	Action	Description				AIR Rule(s)	Action	Description		AIR Rule(s)	
	air:assert :Q air:non-compliant- with :Whatever-Policy ;	Cannot see type:A or type:B, they are restricted. Take action of any pattern matches.			:P s:var :V2. :W s:triplePattern :T2. :T2 log:includes [] type:B :V2	:P s:var :V1. :W s:triplePattern :T1. :T1 log:includes [] type:A :V1	air:assert :Q air:non-compliant- with :Whatever-Policy ;	Cannot see type:A, it is re- stricted.		:P s:var :V1. :W s:triplePattern :T1. :T1 log:includes [] type:A :V1	RETRIEVE Restriction
We can create hybrid policies, i.e. "if using type:A must re- trieve type:B and vice versa".	Assert compliance if first pat- tern is met. Assert non- compliance if any remaining patterns are met. Assert com- pliance if no patterns at all are met.	If retrieving type:A, must also use type:B. All or none.	:P s:var :V2. :W s:triplePattern :T2. :T2 log:includes [] type:B :V2	:P s:var :V1. :W s:triplePattern :T1. :T1 log:includes [] type:A :V1	:P striplePattern :T2. :T2 log:includes [] type:B :V2	:P s:var :V1. :W s:triplePattern :T1. :T1 log:includes [] type:A :V1	None	1		Unsupported.	RETRIEVE Inclusion
We can create hybrid policies, i.e. "may do all but one of: use type:A, retrieve type:B".	Assert non-compliance.	Cannot retrieve type:A and type:B. Generalizes to MAX(N- 1,N)			:r s:var: v.z. :W s:triplePattern :T2. :T2 log:includes [] type:A :V2	:P s:var :V1. :W s:triplePattern :T1. :T1 log:includes [] type:A :V1	None	-		Unsupported.	RETRIEVE Exclusion
Hybrids possible. Can only have one type on the left hand side. Doing an AND is easy, the OR involves combinatorics. We can have mandatory filters us- ing this policy.	If first rule matches, assert non- compliance if any of the follow- ing rules match, and compliance if they all fail.	If retrieve type:A must not re- trieve type:B or type:C.		:P s:var :V3. :W striplePattern :T3. :T3 log:includes [] type:C :V3	:P s:var :V2. :W s:triplePattern :T2. :T2 log:includes [] type:B :V2	:P s:var :V1. :W s:triplePattern :T1. :T1 log:includes [] type:A :V1	If first rule matches, assert non-compliance if second rule matches, and compliance if sec- ond rule fails.	If retrieve type:A, must not re- trieve type:B.	:P s:var :V2. :W s:triplePattern :T2. :T2 log:includes [] type:B :V2	:P stvar :V1. :W striplePattern :T1. :T1 log:includes [] type:A :V1	RETRIEVE Chaining
Hybrids possible. Can simplify to [] type:X [] for "any access". USE or RETRIEVE are special cases.	No distinction between USE and RETRIEVE. See USE ta- ble.						No distinction between USE and RETRIEVE. See USE ta- ble.				RETRIEVE Default Deny

Table 3.4: Automatic policy generation: RETRIEVE.

3.2.4 Query History with check-compliance

This project supports query histories. The history is a record of every query that a user has made, that can be used when checking for compliance with a query. In fact, exclusion is the only policy for which it makes sense to use the query history; moreover, exclusion should always use the query history. It is important to note that the query history is not always helpful. Restriction never needs it, since single queries simply fail. Inclusion does not need it either, since one query must always contain all of the terms. Chaining should leave history disabled, since having used a condition in the past doesn't guarantee that the current query is compliant. Default deny never needs it, since its behavior is so similar to restriction.

We implement query history as follows.

- User calls "check-compliance POLICY NEW-QUERY QUERY-DESC" wrapper script. POLICY is an AIR policy; NEW-QUERY is a query in n3, perhaps the output of the sparql2n3 script, and QUERY-DESC is a file like query-req.n3 that defines the location of the query and history files.
- Wrapper script appends NEW-QUERY to the history file in QUERY-DESC, say, history.n3. This accomplishes three things:
 - we only need to look at QUERY-HISTORY when reasoning, making it much easier to automatically generate policies that only grow linearly with the number of terms.
 - AIR will explicitly know which query is the current (new) query.
 - This guarantees that the QUERY-HISTORY is never empty, eliminating a tricky edge case.
- 3. Wrapper script writes NEW-QUERY to a file, say, query.n3.
- 4. Wrapper script calls policyrunner.py, and returns the output of policyrunner.py for viewing.
- 5. Optional. If policy is non compliant, we can remove it from the history file.

This approach is sufficient, and preserves correct AIR semantics of the query and the history.

3.2.5 Policy Generation User Interface

The template policies defined above allow a user to easily generate policies. In order to do so, a user interacts with the policy generator. Depending on the policy, a user inputs variables to catch, whether to act on a USE or a RETRIEVE, a policy name, a policy description, and if necessary, whether or not the query ought to be history enable.

3.2.6 Compliance Testing and Browser Presentation in Tabulator

Tabulator is a Firefox browser plugin that enhances Semantic Web data. In this project, it performs two functions. One, it organizes the display of a policy and the of the policy runner, in a form that is readable hy a human. Figure 3-4 demonstrates this functionality. Tabulator also makes it easier for a user to understand the reasoning generated, as figure 3-5 demonstrates.

The majority of the features discussed in this section are the work of Oshani Seneviratne, as it falls under the umbrella of her ongoing Tabulator work. Screenshots of the Tabulator UI are in figures 3-4 and 3-5.

3.2.7 Implementation Note

With so many tools written in different languages and offering very specialized functionalities, it was necessary to write some simple tools to tie them all together. The scripts provide additional functionality, such as query conversion from SPARQL to N3, query history functionality, and template-based policy generation. Various scripts power the translator, the compliance checker, the policy generator, and user-facing Web pages. We discuss the functionality and design choices of those wrappers here; their full source is in the appendix.

🕙 Policy-generat	or UI - Mozilla Firefox 📃 🗖 🔀
<u>File Edit V</u> iew	Hi <u>s</u> tory <u>B</u> ookmarks <u>T</u> ools <u>H</u> elp
S> - C	🗙 🏠 📋 http://dig.csail.mit.edu/2009/IARPA-PIR/Yotam_Work/Sampe_Policy_UI/Policy-g 🏠 🔹 🚱
Policy-general	tor UI +
Policy Type List	Restriction Policy Creator A Restriction Policy does not allow a person to use, retrieve, or both use and retrieve a variable with a certain attribute.
<u>Restriction</u>	Policy Name (Optional)
Policy • Inclusion <u>Policy</u> • <u>Exclusion</u> <u>Policy</u>	Please specify a policy name if desired.
Default Deny Policy	Policy Description (Optional)
• <u>Chaining</u> <u>Policy</u>	Please provide a policy description if desired. Namespaces Please provide the namepaces you wish to be used in the policy. Place each URI on its own line. Make sure to separate them with commas, and provide a unique prefix for each. @prefix: chttp://URL1.n2>. @prefix: chttp://URL1.n2>.
	Included Attributes Please list the attributes that may not be included, and whether for retrieval, use, or both. Cannot Use Cannot Retrieve Both Add Another Variable Submit
Done	Submit!

Figure 3-3: A screenshot of the policy generator, courtest of Yotam Aron.

AIR Policy Tutorial -	mit.edu/TAMI/20	http://dig.csail.mit.edu/TAMI 🗶 📑	
MITProx Card Policy	rule type label	MITRule 1 Policy MIT prox-card policy	
MITRule 1	pattern	data D U purpose P1 type Use Event D type Prox Card Event	
	rule type alt	MITRule2 Belief Rule rule MITRule3	
MITRule2	assert description	U compliant with MITProx Card Policy	
	pattern	P1 same As criminal law enforcement	
	assert	U non compliant with MITProx Card Policy	
MITRule3	description pattern	P1 is different from a criminal activity	
	type	Belief Rule	

Figure 3-4: Tabulator browser presentation of the MIT Prox Card policy.

http://m	nr-burns.w3.org/cg	i-bin/server_cgi.py?logFile=http://dig.csail.mit.edu/2009/IARPA-PIR/query1.n	
			(
	· · · · · · · · · · · · · · · · · · ·		
ittp://n	nr-burns.w3.org/	cgi-bin/server_cgi.py?log-lie=http://dig.csail.mit.edu/2009/IARPA-	
K/quer			
×3 XI			
Query I I	s non compliant with :	SSNPOIICY	
Query 1	is non compliant with	SSNPolicy	\supset
More Info	ormation Start Ov	er	
The 1		- CCN, where is the universide and estricture CCN, where	
The	query, Query I, use	es SSN values in the where clause and retrieves SSN values	
remises	1		
WHERE	Triple Pattern	S ssn N	
т	includes	S ssn N	
g0	variable	N	
WHERE	Triple Pattern	S ssn N	
т	includes	S ssn N	
g0	variable	Ν	
WHERE	Triple Pattern		
_	·		
a0	variable	S ssn N	
90	vanable		
pe ▶●R	DFDocument		
nd All			

Figure 3-5: Tabulator justification user interface.

3.3 Summary

This chapter explained the functionality of the system. It can convert SPARQL queries to N3. It defines a set of primitive policy types. It allows a user to easily create queries, possibly using information about an existing database. It can enable logging of queries. It allows a user to check a query against a policy for compliance. Finally, it allows a user to understand the compliance explanation generated by a policy runner. In sum, this is a complete, end to end system for policy assurance, that is ready to be integrated into an RDBMS.

All of the code in this section and the appendix are available in the MIT CSAIL Decentralized Information Group's code repository. Please contact a member of DIG for access.
Chapter 4

Performance

In any practical system, performance is a concern. This project focused on providing a functional implementation of a policy assurance system, and achieved that goal. In terms of performance, the only optimization we performed was to ensure that the query history grows linearly with the number of queries, and that policies grow linearly with the number of items that they check.

It is relatively easy to write policies that grow faster than would be feasible to implement. For example, one of the primitives that we originally sought to implement was a MAX(m,n) rule, meaning, "given n fields, the user may only see up to m of them." This policy is trivial in the case of MAX(n-1,n), in that the only way to trigger it is to access all n fields. However, the current AIR language would require that for a general implementation of MAX(m,n), we check every possible combination. This is due to the fact that AIR is a production rule system and does not maintain any state. Thus, a straightforward rule like MAX(m,n) grows exponentially with (m - n).

With the help of Yotam Aron, we ran a series of tests using the history policy and the AIR reasoner. The time values listed are the times given by the AIR reasoner itself; there may be more accurate ways of measuring run time. We have found that the way we write queries will influence the running time of our checks. The AIR reasoner abhors complexity, so any attempt to limit the patterns that it checks will help it to run faster. The reasoner runs faster with a smaller history file. We can

Items in Query History	Reasoning	After Loading	Actual Reasoning
0	1.43096	1.094741	1.0751231
1	1.52762	1.32497	1.307404
2	1.7949221	1.601938	1.584473133
3	2.3070598	2.0158172	1.99806
4	2.728109121	2.513138056	2.495877981
5	3.490919828	3.255658865	3.238258839
6	4.528033018	4.235877991	4.218565941
7	5.564540863	5.312869072	5.295244932
8	7.044108152	6.777283192	6.759871006
9	8.317480087	8.051841974	8.028425932
10	10.18744802	9.915023088	9.896337986
11	12.22676992	11.94390798	11.925632
12	15.13193297	14.83994699	14.82177997
13	17.82276082	17.52010894	17.502671
14	21.40320492	21.10541201	21.087852
15	24.56065702	24.24677181	24.22919488
16	29.64700389	29.2435019	29.17709184
17	33.9228971	33.52368808	33.4938252
18	44.79428411	44.41020393	44.35875702
19	50.51201296	50.00673199	49.98922586

Table 4.1: Running time, in seconds, of unoptimized queries.

tailor our policies to shrink the size of air:pattern fields at the expense of creating more rules.

Yotam ran one test suite running Ubuntu on VMWare. Resources allocated: 512 MB memory, 12GB hard disk space. The results are in table 4.1.

Yotam then ran one additional test suite running Ubuntu on VMWare. Resources allocated: 768 MB memory, 12GB hard disk space. Despite the difference in memory allocation, this shows a good trend, with better scaling for a large history. The results are in table 4.2.

We can clearly see exponential growth toward the right hand side of these figures. At present, this is the greatest impediment to scaling. Future changes or optimizations to the reasoner could help this run faster.

In practice, our finds show that it would be difficult for a policy to perform any matching against more than about 100 terms at a time. Any policy that does not support query history will likely begin to have difficulty with a perticularly complex



Run Time for Unoptimized Policy

Figure 4-1: Unoptimized policy run time from table 4.1.

Items in Query History	Reasoning	After Loading	Actual Reasoning
0	1.548950911	1.343198061	1.324027061
1	1.335716963	1.097095966	1.078705072
2	1.426616192	1.18595314	1.159878016
3	1.465291023	1.198924065	1.178475857
4	1.604300976	1.294827938	1.275640965
5	1.621567011	1.348763943	1.330348969
10	2.188154936	1.863446951	1.843798876
15	2.896314144	2.549362183	2.529920101
20	3.677877903	3.288823843	3.269448996
25	4.998191118	4.573461056	4.555090189
30	6.7367239	6.265938997	6.247066021
40	12.97512007	12.29570103	12.26716304
50	21.24516296	20.6023891	20.57518101
60	31.52937508	30.80437517	30.78589511
70	47.03233194	46.23697996	46.211133
80	63.81845903	62.74732184	62.72839189
90	93.3954649	92.36639786	92.34837484
100	117.659286	116.265708	116.2480021
200	864.6837819	862.648648	862.6293769

Table 4.2: Running time, in seconds, of optimized queries.



Run time for optimized policy

Figure 4-2: Unoptimized policy run time from table 4.2. Note log scale on the X axis.

query that accesses on the order of 100 different variables. A policy that supports a query history will have difficulty if there are on the order of 100 queries in said history. A more robust machine might bump these figures by an order of magnitude, but these seem to be the limitations of our system with hardware available today. Thus, it is likely that an enterprise environment looking to implement this software will hit its performance limitations relatively quickly, without additional, out-of-system means in place to mitigate its limitations.

Chapter 5

Related and Prior Work

This chapter presents some of the prior methodologies in database security, as well as other uses of Semantic Web technology in this field. There is some overlap between this and preceding chapters.

5.1 Policy Awareness

The Information Accountability paper [10] describes the arenas in which policy-aware systems are if interest. Among these are privacy, copyright, surveillance, and data mining. The paper argues that if we can trace who uses data and how they use it, if we have a way of finding accountability, we can begin to move toward policies that are between full disclosure and full control. The paper offers scenarios of how information aware systems would be useful in everyday life.

REIN [14] (Rei and N3, named after the Rei policy specification language and the Notation 3 language) describes a system offering "policy management". The system is policy-language agnostic, and operates in an independent environment, offering a yes/no opinion of policy compliance to an existing system. The REIN paper defines three possible operating modes. In the server-side rules operating mode, the analogy of choice is an application for a library card. The user does not know the rules, but supplies the information requested on the form to the library for processing. In the client-side rules mode, the onus of rule checking is on the user. The hybrid mode offers a way of sharing this responsibility. At present, REIN is implemented in Python, using the N3 policy language and the CWM reasoner.

There is prior work demonstrating the use of a policy-aware system for checking license information. [15] [16] In this case, the system uses Creative Commons license policies to check for compliance with the content creator's sharing preferences. This system is a way of "keeping honest people honest." The system, as implemented, converts meta data on an image file to an AIR policy, and checks usage patterns against a set of rules that parallel the Creative Commons license policies. If, for example, an image from the Flickr Web site is used on someones personal page, in violation of the content creator's wishes as expressed in their choice of Creative Commons licensing, this tool will detect a conflict.

5.2 Methodologies of Access Control

It is instructive to look at the approaches systems designers take in designing traditional access control systems. Systems designers tend to have a different mind set when approaching security problems than Web designers: systems designers tend to favor closed systems, whereas Web designers tend to favor open systems. Existing relational database systems extend on these technologies in their implementation of access control.

5.2.1 Mandatory and Discretionary Access Control

The traditional models of access control, circa the mid 1980s, consist of mandatory access control, or MAC, and discretionary access control, or DAC. The origins of MAC lie in secure military systems, as a way to enforce permissions on proprietary, secured, confidential data. [7] In an MAC system, the only way to gain access to data is to expressly have an outside authority grant access. There is absolutely no provision for an end user to alter permissions on data. The primitives for enforcing access control extend deep into the design of the operating system. MAC systems explicitly define rings of access. Some secure Unix operating systems, including SELinux, utilize MAC design principles.

Discretionary access control is a design principle that allows a user to show that they have the credentials to access data. [7]Users have the capability of modifying their own permissions, or the permissions of others. A widespread example of a DAC system is that of Unix style permissions. On Unix systems, everything, from devices to network sockets to data on a disk, is represented as a file. Every file has three sets of permissions: those for the owning user, for the owning group, and for everyone else. Each of these sets may be granted (or denied) permissions to read, write, and/or execute a file. The operating system checks the user's credentials and the file's permissions before enabling an operation. Access Control Lists (ACLs) are another example of a DAC system. There is prior work investigating the use of DAC in object-oriented databases. [6] Such a security model works well for general purpose computing, but not in all scenarios.

The excessive rigors in the implementation of a MAC system, and the inadequacy of a DAC system, have led systems researchers to newer design philosophies that allow finer granularity in security settings.

5.2.2 Role Based Access Control

In the early 1990s, systems researchers began to find practical limitations in the expressive power of the discretionary access control model. Researchers presented role based access control as a more secure, easy to implement alternative to discretionary access control. [8] Role based access control is a form of mandatory access control, in that users obtain their permissions from an outside authority. The primary difference is in the structure of permissions. In role based access control, a user can be assigned to one (or more, in some implementations) "roles". As an example, a user in a large company might wear several hats throughout their time in the company, from "human resources associate" to "financial associate" to "systems analyst". Each of these roles requires a different set of permissions. The formal description of role based access control, demonstrated in the paper, shows the power and flexibility of assigning roles rather than individual permissions. It is particularly well suited to

assigning permissions where a high user turnover rate is present.

5.2.3 Rule- and Policy-Based Access Control

Rule based access control is an extension and a generalization of role based access control. Instead of simple roles, we can use rules and logic to derive whether or not a particular agent should have access to a particular resource. As an example, consider a student trying to access a secured Web page. The student may have credentials saying, "I am a graduate student", "my adviser is Professor Smith", and "I am a member of Research Group X". The page may have a set of rules defining access: it may be restricted to individuals who are members of certain research groups, and also graduate students. Different versions of the page may exist based on the credentials used. REIN [14] and SWRL [9] provide sample implementations of policy based access control. This thesis mimics the RBAC approach, using a similar methodology to define and process rules.

5.3 Prior Work in Relational Databases

This project seeks to address a particular shortcoming in database systems at present. In order to enforce access control policies, the database must be able to see the queries that a user is making against the database. This poses a problem when the queries themselves may reveal sensitive data. Access control lists are insufficient in this regard, whereas misuse and intrusion detection efforts are more closely related with our work.

5.3.1 Access Control Lists

Modern relational database management systems utilize some form of discretionary or role-based access control to regulate access to tables, using some form of access control list. The database administrator may restrict database access to particular rows or columns in particular tables, may grant or deny read and write permissions, and may limit the user to executing particular queries. [5] In effect, all of these systems limit the user to seeing a particular subset of the data in the database.

It follows that the structure of the access control policy must closely follow the structure of the data itself. The structure of the security policy may divulge secure information, and may not be tolerant of changes to the underlying structure of the database.

5.3.2 Access Control Features In A Modern RDBMS

Modern RDBMS packages, such as Oracle, offer a wide variety of access control features. We can roughly group them into five categories. [17] [18] [19]

- Privileges
- Views
- Stored Procedures
- Roles
- Virtual Private Databases

Privileges explicitly grant access to perform an action on a particular database object. A database administrator, or a user with appropriate privileges, can grant or revoke privileges. A view is a dynamic table, the output of a particular query against the database. A stored procedure is a particular command, like a user defined function, provided for database access. A role, as described in the preceeding section on role-based access control, is a type of meta-user that helps congregate database permissions. A virtual private database is a form of information hiding, partitioning the database such that a user only sees certain data.

These systems are able to specify access rights for users and groups on databases, and work well enough in the majority of usage cases. However, this design is not without its limitations. The access policies must closely follow the structure of the data, in many cases involving the database administrator to know a great deal about the structure of the tables. Changes to the layout of data will break previously defined policies. Privileges can be granted or revoked in a conflicting fashion, often making it difficult to determine correct access. Views and virtual private databases may be overly restricting, preventing honest individuals from carrying out their responsibilities. Our work seeks to address the numerous limitations with systems of this design by introducing a more flexible, more abstract means for defining permissions.

5.3.3 Misuse and Intrusion Detection

There is prior work in the field of intrusion detection systems in databases. Intrusion detection researchers argue that, though an outsider threat is very real, current databases do little to guard against violations of policy conducted by a trusted insider with database access. Detection of Misuse in Database Systems (DEMIDS) [20] was one of the first systems to detect potential misuse in a database system. By monitoring queries, using audit logs, and determining a frequent item set, the DEMIDS system is able to guess whether or not a particular access by a particular user is likely to be permitted. This approach suffers from a granularity issue: with too high or too low granularity, it is likely that the system will see a high number of false positive policy violations, or respectively, a high number of false negatives.

Later systems expand on some of the DEMIDS ideas, adding better learning or classification techniques. Cathey et. al. [21] developed a system that uses a combination of user query profile learning, abnormal query behavior detection, clustering documents, clustering query results, and relevance feedback to better understand how particular users and groups access a database. Over time, Cathey's system will "learn" common access patterns. Kamra et. al. [22] extend this theme further, requiring the use of role-based access control, and using a naive Bayes classifier to learn standard access patterns. Their approach argues that, by only considering roles instead of individual users or tables, the granularity is correct. Kamra's paper also argues against a formal definition of intrusion detection, saying that the intrusion detection application is much more purpose-built, and therefore less suited to a formal approach. The aforementioned techniques would be helpful in the initial instrumentation of rule sets, by finding common ways that users access data.

5.4 Alteration of Data

An orthogonal approach to security of certain kinds of sensitive data is the alteration of the data itself. If we restrict the utility of the data through alteration, we can, in theory, limit the usefulness of the data, its sensitivity, and its potential for damage. The classic example of this approach is the use of a heavy black marker to manually censor sensitive government documents prior to public release. There has been some research into formalizing and automating this process.

Sweeney has published a great deal of research on this particular topic. kanonymity [23] [24] offers a formal approach to the de-identification process. A data set that offers k-anonymity for some k is constructed such that, for every attribute (column), there are at least k occurrences of that attribute (rows). Phrased differently, a data set of census data would be anonymous for k = 2 if there were at least two occurrences of each name, address, age, and so on. By setting a value of k, dependent on the application, we can guarantee difficulty in re-creating and de-identifying the data set. Datafly [25] is a system that can de-identify data automatically, removing singletons, changing social security numbers (SSNs) and dates of birth, changing ZIP codes, and the like.

De-identification and privacy problems are nothing new to the biomedical informatics world. Access to, and use of, medical data faces heavy regulation, most prominently from the Health Insurance Portability and Accountability Act (HIPAA) of 1996. The Multi-parameter Intelligent Monitoring for Intensive Care database, or MIMIC II (http://mimic.mit.edu/) is a database, provided by an MIT research group, for research purposes. The MIMIC group has published many papers discussing and evaluating different de-identification algorithms.

With access to the database, our system could potentially anonymize data using any of a number of well known techniques. This would allow us to keep original data, and only serve data that is as specific as it needs to be for a particular application.

Chapter 6

Future Directions

This work is a solid beginning for future work in Semantic Web based policy assurance. It is also just that: a beginning. There are some important pieces of this project that need to be addressed, and some interesting applications that are just beginning to scratch the surface of what is possible with our reasoner. In this chapter, we discuss future work related to this project in this chapter. This is just a sample of the directions in which this work may grow.

6.1 SPARQL Endpoint Integration

The policy assurance implementation in this paper serves as a standalone component. As we mentioned in the earlier Policy Assurance section, an important future step is to integrate this work with a SPARQL endpoint to demonstrate compliance with incoming queries in real time. To do this, the endpoint would need to be able to look at the AIR reasoner's output, and pull compliance information from that output. A particular endpoint could perform verification using a local or online reasoner, depending on the application specific needs.

6.2 SQL Support

The majority of databases in production use the SQL query language. In order to be compatible with most existing database implementations, our project needs to be able to handle SQL queries. There are several pieces that need to come together for SQL to be a supported language. First, we would need to find a way to translate SQL queries to N3. The SPASQL project (cite) provides for mapping SPARQL queries to SQL. This translation needs more research attention to determine the challenges for converting SQL, or at least a portion thereof to SPARQL. (A conversation with Eric Prud'hommeaux suggests that a one-pass approach may be insufficient, and that this could be a Ph.D. thesis topic.) Secondly, we need a way of describing the fields in a SQL table using URIs.

6.3 Completing and Porting the N3 Translator

The current SPARQL to N3 translation does not support GRAPH, nor does it completely support the built in functions of SPARQL, or nested queries. Furthermore, the current one-pass approach Instead of a mere re-serialization of the SPARQL query in one pass, it may be necessary to perform additional processing. With more modification to the swobjects code base, multiple pass translation is possible, enabling a much richer translation.

The swobjects approach to query translation was convenient, since the author had easy access to the main developer of swobjects, but it may not be ideal. The majority of the code used in this project, including the reasoner and various scripts, are in Python. The fyzz project [26] implements a SPARQL lexer and parser in Python. This recent project may offer better integration into the rest of the project, allowing a unified query toolkit.

6.4 Policy Generation from Natural Language

Some of Oshani Seneviratne's previous work deals with parsing natural language. It should be possible to parse natural language, or at least a constrained version thereof, in such a way that we could automatically generate policies from that. Using natural language as input would be a great benefit to the usability of this system.

6.5 Semantic Policies

Our collaborators at Lincoln Lab proposed implementing policies which perform reasoning in order to deduce a conclusion. Instead of directly specifying, "do not allow address searches where CITY equals Boston," we can more vaguely specify "do not allow searches in Boston." With an appropriate set of assumptions and data, the reasoner would be able to piece together that Boston contains streets, neighborhoods, ZIP codes, etc. Yotam Aron's work is exploring these possibilities directly, by encoding knowledge bases for the reasoner. A long term goal would be to adapt the reasoner to use existing ontologies in drawing conclusions.

6.6 Database Description

To make things simpler for a user, this project should support loading a database ontology to automatically populate the fields in the policy generation user interface. This reduces duplicated effort, as well as reducing the possibility of simple typographical errors. The database ontology is an N3 description of the attributes in a database, with some pre-defined filters.

Chapter 7

Concluding Thoughts

The problem of policy assurance is one of transparency, accountability, and correctness. The classic approach to sensitive security issues has always been a very blackand-white "all or none": either data is visible, or it is restricted. In a modern and inter-disciplinary world, adding more shades of gray to the possibilities for security is increasingly important. The security approach that we describe in this thesis does not simply ask *who* a user is, but also *what* they are trying to see. At present, SQL database security operates on the level of a table. At best, we can grant or deny access to particular rows and columns of a table, but the limitation is still a table: a flat, two-dimensional structure.

The Semantic Web works with data that comes, not in table form, but in graph form. The Semantic Web is built on top of the Web, with its myriad of interconnected links. By using Semantic Web technologies, we can begin to define policies that act not simply on tables, but on a more complex, more abstract data set. Furthermore, we can define policies that are as fine-grained as individual terms in a query.

By looking at queries, this project takes a different approach than other database security initiatives. This system can operate completely externally to a database, or function as an addition. It can protect sensitive data contained in user queries by design. It provides a human- and machine-readable "paper trail" that gives more explicit information about policies to users and auditors, while allowing enough freedom for administrators to easily create interesting policies. This design gives a performance benefit: by looking at queries and not results, the system can save a heavily loaded database server from executing queries that are not compliant, and would be a use of valuable cycles.

Our approach is particularly valuable in multi-tiered environments containing highly sensitive data, where different parties have different access privileges and different access needs. This approach minimizes the amount of information in a log, to the information that is strictly needed to verify a particular assertion. This improves usability and reduces the risk of data leakage, by showing an auditor everything they need to see, and nothing they don't. This is a boon, as it allows even more levels of access, potentially enabling new workflows.

The specific approach of this project was to check query patterns. We converted SPARQL queries to the N3 language, and provided a tool for doing so. We wrote policies in the AIR language, and provided tools to help an administrator write their own policies according to our boilerplate. We provided a way to check the queries against the policies, using the Firefox Web browser. The system achieves the goal of usability and scalability.

Of course, this system is not without its limitations. The current implementation scales reasonably well for hundreds or thousands of entities, but not for millions of entities. Some important future work will add needed performance and usability improvements.

It is our hope that this system will eventually see utilization in production, and will grow over time. The author can only begin to imagine the possibilities for this system.

"Share and enjoy."

-Douglas Adams, The Hitchhiker's Guide to the Galaxy

Appendix A

Background Technologies

Data on the World Wide Web is readily available to human agents, but notoriously for machines to interpret. The Semantic Web is an ongoing initiative to develop Webbased data systems that are easily readable by both humans and machines. Common tactics for achieving this goal include tagging data, linking data, and writing software to draw conclusions on data.

This project draws heavily on Semantic Web technologies. In this chapter, I discuss the Semantic Web initiative as well as several technologies that were used in this project. This overview is far from exhaustive, as there are a large number of technologies that comprise the Semantic Web vision; rather, this summary is here to explain the suitability of the technologies chosen, as well as to provide an introduction to these technologies for the reader.

A.1 Semantic Web Overview

A.1.1 The Vision

"The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

-Tim Berners-Lee. From [27]



Figure A-1: Semantic Web "layer cake," showing how components relate [2].

In broad terms, the Semantic Web is a set of extensions to the current World Wide Web designed to make it easier for automated systems to interact data, and thus, to make it easier for humans to interact with data. The goal is to provide tools that make it possible to improve data handling in a large, heterogeneous, decentralized environment such as the Web. The current strategy is to provide tools that allow us to describe data, and tools that perform reasoning over data.

The Semantic Web development effort has directly led to a number of technologies [28] that enable data markup, ontology definition, and other useful tasks. These tools extend existing systems (such as the HTML language), and serve as adaptations of other technologies (such as XML).

Developments in the Semantic Web effort have produced a way of naming abstract concepts, a way of describing concepts using other concepts or literals, and a way of finding descriptions and matching patterns. More importantly, these technologies make it much easier for both human and machine agents to read, reason on, and operate upon data. In a world that is ever more interdisciplinary, being able to connect disparate data is a boon.

Semantic Web technologies are moving out of the realm of research and into

commercial production. Adobe offers tools to embed RDF metadata in a number of formats [29]. The Creative Commons license [30] embeds RDF metadata that helps determine usage rights for media. Major Web search engines as of this mid-2009 writing, such as Google, bing, and Wolfram Alpha, employ Semantic Web technologies to help improve the quality of search results. RDF triples are becoming popular in the biomedical and intelligence communities, where collaboration between disciplines is crucial to success.

A.1.2 The URI

The Uniform Resource Identifier, or URI, is the unique technology that enables naming on the Web [31]. A URI may serve as a pointer to a resource, be it a Web page, an e-mail address, the location of a music file, the name of a service. A URI may point to something that does not exist. Regardless of its destination, a URI serves as a unique identifier in the Web world.

A URI enables us to refer to items in a namespace as well. The idea of "URI as a name" is a crucial cornerstone of the Semantic Web. For example, let us consider the following URI:

http://www.example.org/

This URI points to a resource that might be located there. Now, let's create a simple ontology for describing things that I might find at a local grocery store:

```
http://www.example.org/grocery#banana
http://www.example.org/grocery#apple
http://www.example.org/grocery#spaghetti
```

Each of these URIs contains value itself. When I resolve the URI, and fetch the resource to which it points, I may find a textual description of a banana, or a photograph of an apple, or nutritional information about spaghetti. I may find nothing at all. I can capture the design pattern of common prefixes as a name space. Instead of repeating ourselves, let us define the "grocery" namespace as the set of URIs that begin with "http://www.example.org/grocery".

A.1.3 HTML, the HyperText Markup Language, and XML, the eXtensible Markup Language

The Hypertext Markup Language (HTML) serves as the backbone language of the World Wide Web. As a markup language, it provides a standardized way of creating Web documents that users can read with any of a number of programs, including Web browsers. HTML's characteristic property is its ability to encapsulate data with a pre-defined set of tags, easily recognizable with angle brackets. A common tag may look like this:

 Example Home Page

The Extensible Markup Language, or XML, enables an unlimited amount of additional expressivity in the form of arbitrary user-created tags. I can nest XML tags within one another, allowing us to represent arbitrary tree structures in the language. As an example, I can create a grocery list, building upon the previous example.

```
<grocerylist>
<item moreinfo="http://www.example.org/grocery#banana" quantity="5">
Bananas</item>
<item moreinfo="http://www.example.org/grocery#apple" quantity="2">
Apples</item>
</grocerylist>
```

The expressive power of XML is also its weakness: without any standardization, I would have an infinite number of meaningless and unrelated descriptive languages, defeating our goal of human and machine interoperability. An XML schema gives us a way to standardize a description, but even this is not sufficiently robust for machine readability.



Figure A-2: The RDF logo. Its structure suggests the triple pattern. From [3].

Free text	Triple pattern(s)
The population of	{ "Boston", "population", "1E6" }
Boston is one million.	
The Red Line stops at	<pre>{ "Red Line", "station", "Park Street" }</pre>
Park Street.	
The color of the sky is	{ "Sky", "color", "blue" }
blue.	

Table A.1: Expressing natural language as triples.

A.2 RDF

The Resource Description Framework, or RDF, provides an infrastructure for creating metadata using the ides of the URI [3] [32]. RDF builds on XML technologies to provide a framework for using XML to describe data.

At its core, RDF is little more than a formalization of the triple pattern:

{ subject predicate object }

The triple pattern is a representation of a (noun,verb,modifier) sentence construct. These three pieces of information describe relationships between objects. Indeed, this pattern is very similar to the constructs for description available in many spoken and written languages. As an example, I show how triple patterns might represent the following relationships in pseudocode in table A.1.

I can chain triple patterns together to make more complex statements, as I show in table A.2.

From this information, I could conclude that Zach lives in Illinois². This system of description lends itself to automated reasoning, which is precisely why it is suitable for Semantic Web activity. I will discuss tools to perform this kind of analysis shortly.

In practice, there are two additional clarifications that allow definition of triple patterns. First, every subject, predicate, and object must be a URI, a literal, or

Free text	Triple pattern(s)
Jane has a pet dog	{ "Jane", "pet", "Zach" }
named Zach and a	
house in Illinois.	
	{ "Zack", "species", "dog" }
	{ "Jane", "residence", "house" }
	<pre>{ "house", "location", "Illinois" }</pre>
Zach lives in the	<pre>{ "Zach", "residence", "house" }</pre>
house.	

Table A.2: Expressing natural language as triples.

possibly a variable:

<http://example.com/boston> <http://example.com/population> "1,000,000"

Secondly, RDF triples must be serialized (written) in a standard format. The two popular approaches for RDF serialization are RDF/XML and Notation 3, or N3 (cite http://www.w3.org/DesignIssues/Notation3.html). I can express this example in RDF/XML as follows, using abbreviations for the RDF and fictitious "geography" namespaces for convenience. RDF/XML fulfills the need for expression of triple pattern relationships in a standardized, royalty-free, open format.

```
<rdf:RDF
```

In a more general sense, triple patterns enable the expression of nodes and edges of a (directed) graph. Subjects and objects are nodes in the graph, and predicates define the edges.

A.2.1 Notation 3

Notation 3, or N3, is able to express all of RDF/XML, with the goal of being more human readable. For comparison, I show the same triple pattern with an N3 serialization, assuming the same namespaces.

```
@prefix geography: <http://example.com/> .
geography:boston geography:population 1000000 .
```

The N3 serialization, a form of shorthand or "syntactic sugar", is equivalent to the RDF/XML serialization above; tools exist that will transform one into the other. The real strength of N3 - in light of the goals of the Semantic Web - is that it is more human readable.

Notation 3 is more expressive than pure RDF; indeed, the complete RDF 1.0 specification is a subset of N3. The language supports implication and quantifiable variables ("for all" and "for some"). This additional expressive power enables some interesting applications, such as reasoners, rule systems, and query processing.

A.3 OWL

RDF and the URI together can describe the properties that an object has, by using unique names and expressing subject, predicate, object relationships. In order to have multiple parties agree on a standard set of descriptors for objects, and encode restrictions such as "there is exactly one of these" or "this is a subtype of that", another system is necessary. The Web Ontology Language, or OWL, provides a common way to capture these design patterns [33].

OWL offers a way to create ontologies that is more expressive than XML or RDF. Its three dialects, OWL Lite, OWL DL, and OWL Full, offer graded levels of expressive power. OWL allows URIs to have structure, with cardinality, equivalence, and subtyping. OWL can connect different ontologies.

A.4 Tabulator

The Tabulator [34] is a Semantic Web browser. Implemented as a browser plug-in, or possibly as a standalone application, it makes it easier for humans to view and understand RDF data. Tabulator enables searching, outlining, and tabular display of semantic data. It supports add-ins that enable micro-blogging, and help users understand the structure of some specific RDF formats. Tabulator's role in the Semantic Web project is to serve as a crucial bridge to human usability.

A.5 SPARQL

RDF data defines a graph structure. This graph may be more complex than a relational model, which focuses on linked tables of row/column attributes. Thus, for reasons of performance or simplicity, it may be useful to create a new data structure for storing triple patterns. For the purpose of this paper, it is sufficient to assume that there exists a way to store triple patterns.

Data is meaningless unless it has context and an interested party is able to find it. The URI and the triple pattern [35] allow data to have context, using standards and ontologies that many parties may use. The missing piece of the puzzle is SPARQL, a query language for RDF data and triple stores.

From the SPARQL language specification [4]:

"SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs."

Given the N3 triples in figure A-3, and the SPARQL query in figure A-4, a SPARQL query processor might return the output in table A.3. (This example comes

```
<http://example.org/book/book1> \
<http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Figure A-3: A sample RDF data set, using backslash to denote newline. From [4].

```
SELECT ?title
WHERE
{
    <http://example.org/book/book1> \
    <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Figure A-4: A sample SPARQL query. From [4].

from the SPARQL W3C Recommendation [4].)

SPARQL's syntax is reminiscent of SQL, and this is intentional. A SPARQL implementation will find triple patterns that match the triple patterns specified in the WHERE part of a SPARQL query, and bind and return variables in the SELECT part of the query. In this case, the SPARQL engine will look for any objects in triple patterns, such that the subject of the triple pattern is "<http://example.org/book/book1>" and the object of the triple pattern is "<http://eul.org/dc/elements/1.1/title>". Given only the data in figure 1, it finds exactly one triple pattern with this subject and predicate, the triple pattern with the object "SPARQL Tutorial".

The SPARQL language is a robust query language. In short, SPARQL is to RDF graphs as SQL is to relational databases. The introduction of SPARQL provides a much needed tool for querying RDF graphs, using a simple, familiar syntax. It enables joins through multiple match patterns. It allows a user to CONSTRUCT a new RDF graph based on the results of the query, to FILTER data, to mark a certain match as OPTIONAL, and to alter the results using LIMIT and OFFSET directives. A user can ASK if any pattern matches a pattern, or have the SPARQL engine DESCRIBE its data set. A user can even specify our own RDF graphs for a

Variable	Binding
title	"SPARQL Tutorial"

Table A.3: The output of a basic SPARQL query.

query using FROM and FROM NAMED; the SPARQL engine may or may not handle user-specified graphs. (A more thorough discussion of the features of the SPARQL query language follows in a later section, as part of the discussion of this project's query translator.)

A.6 Reasoning

The reasoning systems in this project build on some classic principles of logic and logical programming. A brief overview of those technologies follows.

A.6.1 Forward Chaining

Forward chaining, or modus ponens, is a reasoning method by which a set of rules are used to draw all possible conclusions about a data set. A rule typically has two parts, a left hand side and a right hand side, an "if" clause and a "then" clause, a condition and a consequent. This is also known as a data driven approach, since the rules used and the outcome will vary depending on the data. Of course, the rules determine which possible outcomes a forward chaining system may reach.

The inverse of a forward chaining system is backward chaining. In a backward chaining system, the goal is to find data to support a conclusion. This approach is also called a goal driven approach.

Indeed, the SPARQL language presented above uses forward reasoning to match triple patterns. In the case of figure A-4, there must be a rule, somewhere, whose condition is "if there is a triple pattern with this subject and this predicate", and whose consequent is "then output that same triple's object". The forward chaining approach allows arbitrary conclusions given data and rules; given an unlimited data set, rules could run for a long time, possibly forever.

A.6.2 Production Rule Systems

A production rule system is a computer program that implements a forward chaining reasoner with a set of rules over a data set. The production rules also feature a condition and a consequent. In addition to forming conclusions, the rules in a production system may have side effects, triggering outside events or causing side effects within the system.

A "naive" production rule system will simply apply every rule, repeatedly, until it finds a match. Some optimizations will change the order of the rules, or perhaps give certain rules priority.

A.6.3 The Rete Algorithm

Rete (pronounced REE-tee) is the name of a particular algorithm used for pattern matching in production rule systems [36]. Developed in the late 1970s by Charles Forgy, Rete offers substantial performance improvements to previous production rule systems by using a tree-based approach. The Rete algorithm makes use of some moderately advanced graph theory in its implementation. for our purposes, what is important is that the algorithm scales well.

A.6.4 Semantic Web Application Platform

The Semantic Web Application Platform is the name of a number of technologies developed for the purpose of furthering Semantic Web research [37]. The platform is working to create tools and utilities to enable conversion, extraction, and use of Semantic Web data.

A.6.5 cwm and cwmrete

Cwm is a software suite written to facilitate Semantic Web work [38]. From the Cwm homepage:

"Cwm (pronounced coom) is a general-purpose data processor for the se-

mantic web, somewhat like sed, awk, etc. for text files or XSLT for XML. It is a forward chaining reasoner which can be used for querying, checking, transforming and filtering information. Its core language is RDF, extended to include rules, and it uses RDF/XML or RDF/N3 serializations as required."

The cwm reasoner is the backbone of many Semantic Web projects, including this one. Cwmrete, an implementation of the Rete algorithm in cwm, allows advanced forward chaining, as required by the AIR language.

A.7 AIR

A.7.1 Introduction

The AIR policy language, for "Accountability in RDF" or "AMORD [39] in RDF," is a specification for writing policies and making assertions. The language makes use of Notation 3 (N3), an RDF syntax, for expressing policies. These policies consist of individual rules and patterns that, if matched, will trigger other rules and assertions. AIR implements a system of production rules using forward chaining reasoning.

The current implementation of the AIR reasoner draws on pre-existing technologies. The reasoner presently incorporates a truth maintenance system (TMS). The AIR reasoner also incorporates work from the Rein project [14], which specified policies written in N3, and reasoning using the cwm engine. Rein also introduced distributed policies, and reasoning over such policies in the Semantic Web.

In its current form, the AIR reasoner consists of several components. Rules files, written in N3, specify policies, each with at least one rules, which can make assertions. Log files, also in the N3 format, contain facts upon which I might apply AIR rules. AIR permits multiple log files. Finally, a reasoner, or "policy runner", accepts a rule file and one or more log files as input. The reasoner will continue to apply the rules using either an open-world or a closed-world assumption until there is nothing more to deduce. For the purpose of this project, the reasoner uses a closed-world



Figure A-5: Diagram of the AIR policy language ontology.

assumption so that all assertions depend solely on rules and policies I provide. The reasoner's output, in the N3 format, details the reasoning it performed, including a nested list of the assertions it made.

The AIR ontology serves to define the AIR language, in showing which classes are subclasses of other classes and demonstrating how rules inherit from containers and actions. A diagram of the ontology appears in figure A-5.

A.7.2 A Brief AIR Tutorial

In this section, I demonstrate the features of a relatively simple AIR policy: a slightly amended version of the policy in the MIT Prox Card Scenario [40]. The sample policy is complete, and exemplifies many of the features of the AIR language. The online AIR policy tutorial is a helpful reference here [41]. The full text of this example policy is in the appendix. The main body of this paper will contain some more advanced policies. It bears mentioning that the policies discussed herein are in "version 1" of the AIR language. All of the policies presented later are in "version 1" as well. An updated syntax, "version 2", is current and was introduced on 13 April 2009. I will quickly demonstrate the differences in versions and rationale for the change here, and discuss the need to move to the new version in the Future Work section.

First, any AIR file should declare the air namespace, as AIR policies rely upon the ontology that this namespace defines. The trailing dot is mandatory.

@prefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#> .

An individual policy may declare any namespace as its local namespace, and also specify any number of additional namespaces.

```
@prefix : <http://dig.csail.mit.edu/TAMI/2007/s0/mit-policy#> .
(additional namespaces)
```

The policy file must declare the variables it will use using an @forAll or @for-Some declaration toward the beginning. This is mandatory, as the AIR file has N3 semantics!

@forAll :U, :D, :P1.

An AIR policy file may contain multiple policies; our example only contains policy. I recall that N3 has some reserved keywords, such as a and is a, which refer to concepts in the OWL ontology. For example, a is shorthand for owl:sameAs. (cite http://www.w3.org/TR/owl-ref/#sameAs-def) A policy may call one or more rules in an attempt to forward chain and make an assertion.

:MITProxCardPolicy a air:Policy;

```
air:rule :MITRule1.
```

Note that this is equivalent to:

http://dig.csail.mit.edu/TAMI/2007/s0/mit-policy#:MITProxCardPolicy \
owl:sameAs http://dig.csail.mit.edu/TAMI/2007/amord/air#:Policy;
http://dig.csail.mit.edu/TAMI/2007/s0/mit-policy#:MITProxCardPolicy \
air:rule http://dig.csail.mit.edu/TAMI/2007/s0/mit-policy#:MITRule1.

A policy will point to one or more rules. A rule can contain:

- a description, which is a free text descriptor of the rule and may contain variable references;
- a pattern, which is a set of conditions that must all be true for a rule to activate;
- a label, which is a name for the rule;
- an assert, which is the action the reasoner takes if the pattern is matched;
- an alt, which is the action the reasoner will take if the pattern is not matched.

The first rule merits further investigation:

```
:MITRule1 a air:BeliefRule;
air:label "MIT prox-card policy";
air:pattern {
    :U a air:UseEvent;
        air:data :D;
        air:purpose :P1.
        :D a mit:ProxCardEvent.
    };
    air:rule :MITRule2.
```

This code captures the following meaning: "Suppose I can find something, call it :U, that is an air:UseEvent, with an air:Data of something else that I can call :D, and an air:purpose of yet another thing that I can call :P1. If that :D turns out to be an mit:ProxCardEvent, then activate :MITRule2."

Let's take a look at the following two rules.

```
:MITRule2 a air:BeliefRule;
```

```
air:pattern { :P1 :sameAs pur:criminal-law-enforcement. };
air:description (:P1 " is same as a criminal activity");
air:assert { :U air:compliant-with :MITProxCardPolicy. };
```

air:alt [air:rule :MITRule3].

```
:MITRule3 a air:BeliefRule;
```

```
air:pattern {};
air:description (:P1 " is different from a criminal activity");
air:assert { :U air:non-compliant-with :MITProxCardPolicy }.
```

This pair of rules says, "If that purpose :P1 I found is, in fact, the same as pur:criminal-law-enforcement, then this :U is compliant. Otherwise, this :U is not compliant!"

The MIT Prox Card Policy, though relatively simple, serves to encode a real-world policy in an electronic format that a machine can readily use. There is some effort to further this work, to encode more real world laws and policies; it is the focus of TAMI, the Transparent Accountable Datamining Initiative (cite tami), and other works (cite any that you find).

A.7.3 Changes to the AIR language

On 15 April 2009, Ian Jacobi announced the following changes to the AIR language specification, bringing it to version 2. As all of the policies in this thesis use version 1 of the AIR language specification, porting them to version 2 is an important future work. The changes are as follows.

1. The air:pattern and air:alt properties have been effectively renamed to air:if and —air:else—. So:

```
[ air:pattern :A ;
  air:alt :B ] .
```

Would become:

```
[ air:if :A ;
air:else :B ] .
```

2. The air:assert property now behaves like the air:assertion property, while air:assertion (and the old functionality of air:assert) is no longer supported. Thus:

```
[ air:assert :A ;
air:assertion :B ] .
```

Would become:

[air:assert [air:statement :A] ; air:assert :B] .

3. The air:rule and air:assert properties are no longer permitted to accept air:Abstract-rule as a subject. They now take air:Abstract-action (like the bnode object of the old air:alt). To allow for the old actions, a new property has been added, air:then, that behaves like air:alt, except that the actions trigger on match. An air:Abstract-action may have only one of air:rule, air:goal-rule, air:assert, air:assert-goal.

In summary, then:

```
[ air:pattern :A ;
  air:rule :B ;
  air:goal-rule :C ;
  air:assert :D ;
  air:assert-goal :E ;
  air:assertion :F ;
  air:alt :AnAlt ] .
```

```
:AnAlt [ air:rule :G ;
air:assert :H ] .
```

Would now become:

```
[ air:if :A ;
air:then [ air:rule :B ] ,
       [ air:goal-rule :C ] ,
       [ air:assert [ air:statement :D ] ] ,
       [ air:assert-goal [ air:statement :E ] ] ,
       [ air:assert :F ] ;
    # Note that the air:alt no longer can hold both air:rule :G
    # and air:assert :H.
    # They must be added as objects of separate air:else statements now.
    air:else [ air:rule :G ] ,
       [ air:assert [ air:statement :H ] ] ] .
```

A.7.4 AIR Summary

The AIR language allows a user to write arbitrary policies, to draw arbitrary conclusions. It is a flexible, relatively easy to use and understand language. It draws upon some technologies that are decades old to perform its reasoning. It provides an interesting application of Semantic Web technologies and is a crucial cornerstone of this project.

A.8 Summary

The ongoing Semantic Web initiative has grown to include a number of technologies, such as RDF and SPARQL, for describing and manipulating data. This project makes use of a number of technologies, most importantly SPARQL and AIR. All of these technologies are fields of active research, and as such, subject to change in the future.
Appendix B

Supporting Code

B.1 MIT Prox Card Policy

```
1 # MIT Prox Card Policy: http://web.mit.edu/mitcard/privacy.html
2 # We use a simplified policy: prox card data can only be used for
3 # criminal investigation.
4
5 #### Namespaces ####
6
7 # The default namespace is this document.
8 @prefix : <http://dig.csail.mit.edu/TAMI/2007/s0/mit-policy#> .
9
10~ # "prox:" is an alias for ":" to improve printed output.
11 @prefix prox: <#> .
12
13\, # AIR (AMORD in RDF) is the policy language.
14 Oprefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#> .
15
16 # Purposes are drawn from a predefined set.
17 @prefix pur: <http://dig.csail.mit.edu/TAMI/2006/s4/purposes#> .
18
19 # The "mit:" namespace defines background terms specific to MIT.
20 @prefix mit: <http://dig.csail.mit.edu/TAMI/2007/s0/university#> .
21
22 @prefix owl: <http://www.w3.org/2002/07/owl#>.
23
24 #### Policy ####
25
26 @forAll :U, :D, :P1.
27
28 :MITProxCardPolicy a air:Policy;
29
        air:rule :MITRule1.
30
31 :MITRule1 a air:BeliefRule;
32
       air:label "MIT prox-card policy";
```

```
33
       air:pattern {
34
           :U a air:UseEvent;
35
            air:data :D;
36
           air:purpose :P1.
37
           :D a mit:ProxCardEvent.
38
       };
39
       air:rule :MITRule2.
40
41
   :MITRule2 a air:BeliefRule;
42
       air:pattern { :P1 :sameAs pur:criminal-law-enforcement. };
43
       air:description (:P1 " is same as a criminal activity");
44
       air:assert { :U air:compliant-with :MITProxCardPolicy. };
45
       air:alt [ air:rule :MITRule3 ].
46
47
   :MITRule3 a air:BeliefRule;
48
       air:pattern {};
       air:description (:P1 " is different from a criminal activity");
49
50
       air:assert { :U air:non-compliant-with :MITProxCardPolicy }.
51
52 #ends
```

B.2 SSN Policy - Original Ontology

```
1 # $Date: 2009-04-13 18:57:50 -0400 (Mon, 13 Apr 2009) $
2 # $Revision: 25833 $
3 # $Author: jsoltren $
4
5 # SSN Policy: No-one can access SSN information from the database
6 # specifically, you cannot query for someone's SSN number
7 # or use it in the filter clause
8
9 Cprefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#> .
10 Oprefix owl: <http://www.w3.org/2002/07/owl#>.
11 Oprefix math: <http://www.w3.org/2000/10/swap/math#>.
12 Oprefix log: <http://www.w3.org/2000/10/swap/log#>.
13 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
14 @prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
15 @prefix : <http://dig.csail.mit.edu/2009/IARPA-PIR/ssn-policy#> .
16
17 @forAll :Q, :P, :W, :V, :T, :F, :S.
18 @forAll :X, :Y, :Z, :A, :B, :O.
19
20 :SSNPolicy a air:Policy;
21
       air:label "SSN sample policy for IARPA PIR project";
22
       air:rule :SSN_RULE1.
23
24 :SSN_RULE1 a air:BeliefRule;
25
          air:label "SSN policy rule1";
26
          air:pattern {
27
              :Q a s:Select;
28
              s:POSList :P;
29
              s:WhereClause :W.
30
          };
```

```
31
           air:description (:Q " is a SPARQL query");
32
           air:rule :SSN_RULE2.
33
34
  :SSN_RULE2 a air:BeliefRule;
35
          air:label "SSN policy rule2";
36
          air:pattern {
37
                 :P s:variable :V.
38
                 :W s:TriplePattern :T.
39
                 :T log:includes { :X <http://xmlns.com/foaf/0.1/ssn> :V
                     }.
                :W s:Filter :F.
40
41
                 :F s:TriplePattern :S.
42
                 :S log:includes { :V [] [] }.
43
           };
44
       #air:description ("The query, " :Q ", uses SSN values in the
          where clause as a pattern, " :T ", and as a filter, " :S ",
          and retrieves SSN values as variable," :V );
       air:description ("The query, " :Q ", uses SSN values in the
45
          where clause, as a filter and retrieves SSN values as well");
           air:assert { :Q air:non-compliant-with :SSNPolicy };
46
           air:alt [ air:rule :SSN_RULE3 ].
47
48
   :SSN_RULE3 a air:BeliefRule;
49
50
          air:label "SSN policy rule3";
51
          air:pattern {
52
                 :P s:variable :V.
53
                 :W s:TriplePattern :T.
54
                 :T log:includes { :X <http://xmlns.com/foaf/0.1/ssn> :V
                     }
55
           };
56
       #air:description ("The query, " :Q ", uses SSN values in the
          where clause as a pattern, " :T ", and retrieves SSN values
          as variable," :V );
57
       air:description ("The query, " :Q ", uses SSN values in the
          where clause and retrieves SSN values");
58
           air:assert { :Q air:non-compliant-with :SSNPolicy };
           air:alt [ air:rule :SSN_RULE5 ].
59
60
61
   :SSN_RULE4 a air:BeliefRule;
62
          air:label "SSN policy rule4";
63
          air:pattern {
64
                 :P s:variable :V.
65
                 :W s:TriplePattern :T.
                 :T log:includes { :X <http://xmlns.com/foaf/0.1/ssn> :Y
66
                     }
67
           };
68
           #air:description ("The query, " :Q ", includes reference to
                SSN number in the where clause as a pattern," :T );
           air:description ("The query, " :Q ", includes reference to
69
               SSN number in the where clause");
           air:assert { :Q air:non-compliant-with :SSNPolicy };
70
           air:alt [ air:rule :SSN_RULE7 ].
71
72
73 :SSN_RULE5 a air:BeliefRule;
```

```
74
           air:label "SSN policy rule5";
75
           air:pattern {
76
                  :W s:OptionalGraphPattern :O.
77
                  :0 s:TriplePattern :T.
                  :T log:includes { [] <http://xmlns.com/foaf/0.1/ssn> []
78
                      }
79
            };
80
            #air:description ("The query, " :Q ", includes reference to
                 SSN number in the optional part of the where clause as a
                pattern," :T );
            air:description ("The query, " :Q ", includes reference to
81
               SSN number in the optional part of the where clause");
82
            air:assert { :Q air:non-compliant-with :SSNPolicy };
83
            air:alt [ air:rule :SSN_RULE4 ].
84
85
   :SSN_RULE6 a air:BeliefRule;
86
           air:label "SSN policy rule6";
87
           air:pattern {
88
                  :W s:TriplePattern :T.
89
                  :T log:notIncludes { [] <http://xmlns.com/foaf/0.1/ssn>
                      [] }
90
            };
            air:description ("The query, " :Q ", does not includes
91
                reference to SSN number in the where clause");
92
            #air:assert { :Q air:compliant-with :SSNPolicy }.
            air:rule :SSN_RULE7.
93
94
95
   :SSN_RULE7 a air:BeliefRule;
           air:label "SSN policy rule7";
96
97
           air:pattern {
98
                  :W s:OptionalGraphPattern :O.
99
           };
            air:description ("The query, " :Q ", has an optional part")
100
101
            air:rule :SSN_RULE8;
102
            air:alt :SSN_RULE9.
103
104 :SSN_RULE9 a air:BeliefRule;
           air:pattern {};
105
106
           air:assert { :Q air:compliant-with :SSNPolicy };
           air:description ("The query, " :Q ", does not have a
107
              reference to SSN number in its optional section as it does
               not have an optional section").
108
109
110 :SSN_RULE8 a air:BeliefRule;
111
           air:label "SSN policy rule8";
112
           air:pattern {
113
                  :0 s:TriplePattern :A.
114
                  :A log:notIncludes { [] <http://xmlns.com/foaf/0.1/ssn>
                      [] }
115
116
           };
```

```
117
            air:description ("The query, " :Q ", does not includes
                reference to SSN number in the optional section of the
                where clases");
118
            air:assert { :Q air:compliant-with :SSNPolicy }.
119
120
121
122
   :SSNPolicy_WhereClause a air:Policy;
123
124
        air:label "SSN where clause policy for IARPA PIR project";
125
        air:rule :SSN_WC01.
126
127
   :SSN_WC01 a air:BeliefRule;
128
           air:label "SSN where clause rule 01";
129
           air:pattern {
130
               :Q a s:Select;
131
               s:POSList :P:
132
               s:WhereClause :W.
133
           };
134
            air:description (:Q " is a SPARQL query");
            air:rule :SSN_WC02.
135
136
   :SSN_WCO2 a air:BeliefRule;
137
138
           air:label "SSN where clause rule 02";
139
           air:pattern {
                  :P s:variable :V.
140
141
                  :W s:TriplePattern :T.
142
                  :T log:includes { :X <http://xmlns.com/foaf/0.1/ssn> :Y
                      }
143
            };
144
            air:description ("The query, " :Q ", includes reference to
                SSN number in the where clause");
            air:assert { :Q air:non-compliant-with :
145
                SSNPolicy_WhereClause }.
146
147
148
   :SSNPolicy_OptionalClause a air:Policy;
149
        air:label "SSN optional clause policy for IARPA PIR project";
150
        air:rule :SSN_OP01.
151
152
   :SSN_OP01 a air:BeliefRule;
153
           air:label "SSN optional clause rule 01";
154
           air:pattern {
               :Q a s:Select;
155
156
               s:POSList :P;
               s:WhereClause :W.
157
158
           };
            air:description (:Q " is a SPARQL query");
159
160
            air:rule :SSN_OP02.
161
162 :SSN_OPO2 a air:BeliefRule;
163
           air:label "SSN optional clause rule 02";
164
           air:pattern {
165
                  :W s:OptionalGraphPattern :O.
```

```
166
                  :0 s:TriplePattern :T.
167
                  :T log:includes { [] <http://xmlns.com/foaf/0.1/ssn> []
                      }
168
            };
            air:description ("The query, " :Q ", includes reference to
169
                SSN number in the optional part of the where clause");
170
            air:assert { :Q air:non-compliant-with :
                SSNPolicy_OptionalClause }.
171
172
173
   :SSNPolicy_OutputRule a air:Policy;
174
        air:label "SSN output rule policy for IARPA PIR project";
175
        air:rule :SSN_OR01.
176
177
   :SSN_ORO1 a air:BeliefRule;
178
           air:label "SSN output rule 01";
179
           air:pattern {
                :Q a s:Select;
180
181
               s:POSList :P;
182
               s:WhereClause :W.
           };
183
184
            air:description (:Q " is a SPARQL query");
            air:rule :SSN_OR02.
185
186
187
   :SSN_ORO2 a air:BeliefRule;
           air:label "SSN output rule 02";
188
189
           air:pattern {
190
                  :P s:variable :V.
191
                  :W s:TriplePattern :T.
192
                  :T log:includes { [] <http://xmlns.com/foaf/0.1/ssn> :V
                      }.
193
            };
            air:description ("The query, " :Q ", tries to output a
194
                variable " :P " that is an SSN");
195
            air:assert { : Q air:non-compliant-with :SSNPolicy_OutputRule
                 }.
196
197
198
199
   :SSNPolicy_FilterRule a air:Policy;
200
        air:label "SSN filter rule policy for IARPA PIR project";
201
        air:rule :SSN_FR01.
202
203 :SSN_FR01 a air:BeliefRule;
           air:label "SSN filter rule 01";
204
205
           air:pattern {
206
               :Q a s:Select;
207
               s:POSList :P;
208
               s:WhereClause :W.
209
           };
210
            air:description (:Q " is a SPARQL query");
211
            air:rule :SSN_FR02.
212
213 :SSN_FR02 a air:BeliefRule;
```

```
214
           air:label "SSN filter rule 02";
215
           air:pattern {
216
                  :P s:variable :V.
217
                  :W s:TriplePattern :T.
                  :T log:includes { :X <http://xmlns.com/foaf/0.1/ssn> :V
218
                      }.
219
                  :W s:Filter :F.
220
                  :F s:TriplePattern :S.
221
                  :S log:includes { :V [] [] }.
222
            };
223
            air:description ("The query, " :Q ", filters on SSN
                variables");
224
            air:assert { : Q air:non-compliant-with :SSNPolicy_FilterRule
                 }.
225
```

226 #ends

B.3 SSN Policy - Current Ontology

```
1 # SSN Policy: No one can access SSN information from the database.
2 # As a result, no one may USE or RETRIEVE data tagged as SSN data.
3\, # We include two separate policies: one for usage, and one for
      retrieval.
4
5 Oprefix type: <http://dig.csail.mit.edu/2009/IARPA-PIR/generic#> .
6 Oprefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#> .
7 @prefix log: <http://www.w3.org/2000/10/swap/log#>.
8 @prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
9 @prefix : <http://dig.csail.mit.edu/2009/IARPA-PIR/ssn-policy#> .
10
11 @forAll :P, :Q, :T, :U, :V, :W.
12
13 # The USE policy.
14
15
  :SSN-USE-Policy a air:Policy;
16
       air:label "SSN USE policy. For IARPA PIR project.";
17
       air:rule :SSN_USE_RULE1.
18
19 :SSN_USE_RULE1 a air:BeliefRule;
20
          air: label "SSN USE policy rule 1. Checks for query.";
21
          air:pattern {
22
              :Q a s:SPARQLQuery;
23
                 s:retrieve :P; - a query doesn't need to output
              #
                  anything in order
24
              #
                                 to USE something!
25
                 s:clause :W.
26
          };
27
           air:description (:Q " is a query.");
28
           air:rule :SSN_USE_RULE2.
29
30 :SSN_USE_RULE2 a air:BeliefRule;
31
          air:label "SSN USE policy rule 2. Checks for use, i.e.
              filtering.";
```

```
32
          air:pattern {
33
                  # :P s:var :V. - we don't need to bind to an output
                     variable
                                         in order to USE an SSN.
34
                  #
35
                 :W s:triplePattern :T.
                 :T log:includes { [] type:SSN :V }.
36
37
                 :W s:triplePattern :U.
                 :U log:includes { :V [] [] }.
38
39
           };
40
          air:description ("The query, " :Q ", includes a USE of an
              SSN in the pattern " :U ". The policy states that queries
              may not USE social security number data.");
41
          air:assert { :Q air:non-compliant-with :SSN-USE-Policy };
42
          air:alt [ air:assert { :Q air:compliant-with :SSN-USE-Policy
              }].
43
44
45
  # The RETRIEVE policy.
46
   :SSN-RETRIEVE-Policy a air:Policy;
47
       air: label "SSN RETRIEVE policy. For IARPA PIR project";
48
       air:rule :SSN_RETRIEVE_RULE1.
49
50
51
  :SSN_RETRIEVE_RULE1 a air:BeliefRule;
52
          air:label "SSN RETRIEVE policy rule 1. Checks for query.";
53
          air:pattern {
54
               :Q a s:SPARQLQuery;
55
              s:retrieve :P;
56
              s:clause :W.
57
          };
58
          air:description (:Q " is a query.");
          air:rule :SSN_RETRIEVE_RULE2.
59
60
61
  :SSN_RETRIEVE_RULE2 a air:BeliefRule;
62
          air: label "SSN RETRIEVE policy rule 2. Checks for retrieval,
              i.e. output.";
63
          air:pattern {
64
                 :P s:var :V.
65
                 :W s:triplePattern :T.
66
                 :T log:includes { [] type:SSN :V }.
67
           };
68
          air:description ("The query, " :Q ", will RETRIEVE SSN data
              in the pattern " :T ". The policy states that queries may
              not RETRIEVE social security number data.");
69
          air:assert { :Q air:non-compliant-with :SSN-RETRIEVE-Policy
              };
          air:alt [ air:assert { :Q air:compliant-with :SSN-RETRIEVE-
70
              Policy } ].
71
72
73 #ends
```

B.4 A sample SPARQL Query

```
1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3 SELECT ?s ?id ?n WHERE {
4    ?s foaf:ssn ?n.
5    ?s foaf:age ?a.
6    ?s foaf:openid ?id.
7    FILTER (?a > 18)
8 }
```

B.5 Abstract SPARQL to N3 Ontology

```
# $Date: 2009-07-27 13:04:49 -0400 (Mon, 27 Jul 2009) $
1
2 # $Revision: 26843 $
3 # $Author: yyyaron $
4
                 <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5 Oprefix rdf:
  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
6
   @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
7
8
9
  @prefix : <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
10
11
  # abstract ontology to express subset of SPARQL queries
12
13
  <> rdfs:comment "abstract ontology to express subset of SPARQL
      queries".
14
15 #example sparql query
16 #PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
17 #PREFIX foaf: <http://xmlns.com/foaf/0.1/>
18 #PREFIX example: <http://example.org/>
19 #SELECT * WHERE {
20 # ?s example:age ?a.
21 #
      ?s foaf:openid ?id.
22 #
      OPTIONAL { ?s example:ssn ?n }.
      FILTER (?a > 18)
23 #
24 #}
25\, # translated to
26 #@prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/abstract-sparql
      #> .
27 #
28 #:Query-1052391960 a s:SPARQLQuery;
29 #
       s:retrieve [ s:var :id; s:var :n; s:var :s; s:var :a ];
30 #
       s:clause [
31 #
          s:triplePattern { :s <http://example.org/age> :a };
32 #
          s:triplePattern { :s <http://xmlns.com/foaf/0.1/openid> :id
      };
33 #
          s:triplePattern { :s <http://example.org/ssn> :n };
          s:triplePattern { :a s:booleanGT "18 "}
34 #
35 #
        ].
36
37 :SPARQLQuery a rdfs:Class;
```

```
38
       rdfs:label "Abstract class of SPARQL queries.".
39
40
  :source a rdf:Property;
       rdfs:label "URIs from which a query may pull data.";
41
42
       rdfs:domain :SPARQLQuery;
43
44 :retrieve a rdf:Property;
45
       rdfs:label "Values retrieved by a query.";
       rdfs:domain :SPARQLQuery;
46
47
       rdfs:range :RetrievedVar.
48
49 :RetrievedVar a rdfs:Class;
50
       rdfs:label "Class consisting of retrieved variables.".
51
52 :var a rdf:Property;
       rdfs:label "Values retrieved by a query.";
53
54
       rdfs:domain :RetrievedVar.
55
56
  :clause a rdf:Property;
57
       rdfs:label "Conditions or where clause associated with the query
           . " ;
       rdfs:domain :SPARQLQuery;
58
59
       rdf:range :Clause.
60
61 :Clause a rdfs:Class;
62
       rdfs:label "Class consisting of triple patterns.".
63
64 :triplePattern a rdf:Property;
       rdfs:label "A triple pattern for each condition/where.";
65
66
       rdfs:domain :Clause.
67
68 # These are arithmetic and Boolean operators that the translation
      recognizes.
69
70 rdf:resourceNegation a rdf:Property;
71
       rdfs:label "Negative of a number, -.";
       rdfs:domain rdf:resource.
72
73
74 rdf:resourceInverse a rdf:Property;
75
       rdfs:label "Inverse of a number, /.";
       rdfs:domain rdf:resource.
76
77
78 rdf:resourceSum a rdf:Property;
       rdfs:label "Sum of two numbers, +.";
79
80
       rdfs:domain rdf:resource.
81
82 rdf:resourceProduct a rdf:Property;
       rdfs:label "Multiplication of two numbers, *.";
83
       rdfs:domain rdf:resource.
84
85
86 :booleanNOT a rdf:Property;
87
       rdfs:label "The Boolean NOT operator, ! or ~.";
88
       rdfs:domain rdf:resource.
89
```

```
90 :booleanAND a rdf:Property;
        rdfs:label "The Boolean AND operator, &&.";
91
92
        rdfs:domain rdf:resource.
93
94
   :booleanOR a rdf:Property;
        rdfs:label "The Boolean OR operator, ||.";
95
96
        rdfs:domain rdf:resource.
97
98
   :booleanEQ a rdf:Property;
99
        rdfs:label "An equal to relationship, =.";
        rdfs:domain rdf:resource.
100
101
102 :booleanNE a rdf:Property;
103
        rdfs:label "A not equal to relationship, !=.";
        rdfs:domain rdf:resource.
104
105
106 :booleanLT a rdf:Property;
        rdfs:label "A less than relationship, <.";
107
108
        rdfs:domain rdf:resource.
109
110 :booleanGT a rdf:Property;
        rdfs:label "A greater than relationship, >.";
111
        rdfs:domain rdf:resource.
112
113
114 :booleanLE a rdf:Property;
        rdfs:label "A less than or equal to relationship, <=.";
115
        rdfs:domain rdf:resource.
116
117
118 :booleanGE a rdf:Property;
119
        rdfs:label "A greater than or equal to relationship, >=.";
120
        rdfs:domain rdf:resource.
121
122 #ends
```

B.6 Sample Restriction Policy

```
1 # Auto-generated AIR restriction policy.
2
3 Cprefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#>.
4 @prefix log: <http://www.w3.org/2000/10/swap/log#>.
5 @prefix math: <http://www.w3.org/2000/10/swap/math#>.
6 @prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#>.
7 @prefix : <>.
8
9 # User-defined namespaces:
10 @prefix example: <http://www.example.com/#> .
11
12 @forAll :P, :Q, :T, :U, :V, :W.
13
14
  :sample-restriction a air:Policy;
15
       air:label "A sample auto-generated restriction policy.";
16
       air:rule :sample-restriction-check.
17
```

```
:sample-restriction-check a air:BeliefRule;
18
19
       air:label "sample-restriction query check";
20
       air:pattern {
21
           :Q a s:SPARQLQuery;
22
              s:retrieve :P;
23
              s:clause :W.
24
       };
25
       air:description ("Make sure this is a query.");
26
       # Goes to the first rule. If there are no attributes,
27
       # simply go to the default rule.
28
       air:rule :sample-restriction-rule-0.
29
30
   :sample-restriction-rule-0 a air:BeliefRule;
31
       air:label "sample-restriction example:name restriction";
32
       air:pattern {
33
       # if RETRIEVE, or alternately, not USE
34
           :P s:var :V;
35
            :W s:triplePattern :T;
36
           :T log:includes { [] example:name :V }
37
           };
       air:description ("A restriction on the RETRIEVE of example:name,
38
            violated by " :T );
       air:assert{:Q air:non-compliant-with :sample-restriction)};
39
40
       # There are more policies, go to the next policy.
41
       air:alt [ air:rule :sample-restriction-rule-1 ] .
42
43
   :sample-restriction-rule-1 a air:BeliefRule;
44
       air:label "sample-restriction example:age restriction";
45
       air:pattern {
46
       # if USE
47
            :W s:triplePattern :T;
48
            :T log:includes { [] example:age :V };
            :W s:triplePattern :U;
49
50
           :U log:includes { :V [] [] }.
51
           };
52
       air:description ("A restriction on the USE of example:age,
           violated by " :T );
       air:assert{:Q air:non-compliant-with :sample-restriction)};
53
       # There are more policies, go to the next policy.
54
55
       air:alt [ air:rule :sample-restriction-rule-2 ] .
56
57
   :sample-restriction-rule-2 a air:BeliefRule;
58
       air:label "sample-restriction example:dob restriction";
59
       air:pattern {
60
       # if RETRIEVE, or alternately, not USE
61
            :P s:var :V;
62
            :W s:triplePattern :T;
63
            :T log:includes { [] example:dob :V }
64
           };
       air:description ("A restriction on the RETRIEVE of example:dob,
65
          violated by " :T );
66
       air:assert{:Q air:non-compliant-with :sample-restriction)};
67
       # There are more policies, go to the next policy.
       air:alt [ air:rule :sample-restriction-rule-3 ] .
68
```

```
69
70
   :sample-restriction-rule-3 a air:BeliefRule;
71
       air:label "sample-restriction example:ssn restriction";
72
       air:pattern {
73
       # if USE
74
           :W s:triplePattern :T;
75
           :T log:includes { [] example:ssn :V };
76
            :W s:triplePattern :U;
77
           :U log:includes { :V [] [] }.
78
           };
79
       air:description ("A restriction on the USE of example:ssn,
          violated by " :T );
80
       air:assert{:Q air:non-compliant-with :sample-restriction)};
81
       # No more policies, go to the base case.
82
           air:alt [ air:rule :sample-restriction-default ] .
83
84
   :sample-restriction-default a air:BeliefRule;
85
       air:label "sample-restriction default rule";
86
       air:pattern { # Empty search.
87
            };
       air:description ("No restricted attributes found.");
88
89
       air:assert{ :Q air:compliant-with :sample-restriction } .
90
91 #ends
```

B.7 Sample Inclusion Policy

```
1 # Auto-generated AIR inclusion policy.
2
3 Cprefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#>.
4 @prefix log: <http://www.w3.org/2000/10/swap/log#>.
5 @prefix math: <http://www.w3.org/2000/10/swap/math#>.
7 @prefix : <>.
8
9 # User-defined namespaces:
10 Oprefix example: <http://www.example.com/#> .
11
12 @forAll :P, :Q, :T, :U, :V, :W.
13 @forAll :TO, :UO, :VO. # Variables for example:name
14 @forAll :T1, :U1, :V1. # Variables for example:age
15 @forAll :T2, :U2, :V2. \# Variables for example:dob
16 @forAll :T3, :U3, :V3. # Variables for example:ssn
17
18
  :sample-inclusion a air:Policy;
19
       air:label "A sample auto-generated inclusion policy.";
20
       air:rule :sample-inclusion-check.
21
22
  :sample-inclusion-check a air:BeliefRule;
23
      air:label "sample-inclusion query check";
24
       air:pattern {
25
          :Q a s:SPARQLQuery;
26
             s:retrieve :P;
```

```
27
              s:clause :W.
28
       };
29
       air:description ("Make sure this is a query.");
       # Goes to the first rule. If there are no attributes,
30
31
       # simply go to the default rule.
32
       air:rule :sample-inclusion-compliance-rule.
33
34
   :sample-inclusion-compliance-rule a air:BeliefRule;
35
       air:label "sample-inclusion, an inclusion policy.";
36
       air:pattern {
37
           # Catch RETRIEVE
38
           :P s:var :VO.
39
           :W s:triplePattern :TO.
40
           :TO log:includes { [] example:name :VO } .
41
           # Catch USE
42
           :W s:triplePattern :T1.
43
           :T1 log:includes { [] example:age :V1 }.
44
           :W s:triplePattern :U1.
45
           :U1 log:includes { :V1 [] [] } .
           # Catch RETRIEVE
46
           :P s:var :V2.
47
48
           :W s:triplePattern :T2.
           :T2 log:includes { [] example:dob :V2 } .
49
50
           # Catch USE
51
           :W s:triplePattern :T3.
           :T3 log:includes { [] example:ssn :V3 }.
52
           :W s:triplePattern :U3.
53
54
           :U3 log:includes { :V3 [] [] } .
55
           };
56
       air:description ("This inclusion policy states that you must do
           all four of these actions: RETRIEVE example:name, USE example
           :age, RETRIEVE example:dob, USE example:ssn. This query is
           compliant: " :Q ) ;
57
       air:assert{ :Q air:compliant-with :sample-inclusion } ;
58
       # All or none. If we don't match all, start looking for
           individuals.
59
       air:alt [ air:rule :sample-inclusion-rule-0 ] .
60
61
   :sample-inclusion-rule-0 a air:BeliefRule;
62
       air:label "sample-inclusion example:name usage check";
63
       air:pattern {
64
       # if RETRIEVE, or alternately, not USE
65
            :P s:var :V;
66
           :W s:triplePattern :T;
67
           :T log:includes { [] example:name :V }
68
           };
       air:description ("Inclusion includes RETRIEVE of example:name,
69
          violated by " :T );
       air:assert{:Q air:non-compliant-with :sample-inclusion)};
70
71
       # There are more attributes, go to the next policy.
72
       air:alt [ air:rule :sample-inclusion-rule-1 ] .
73
74
   :sample-inclusion-rule-1 a air:BeliefRule;
75
       air:label "sample-inclusion example:age usage check";
```

```
76
        air:pattern {
77
        # if USE
78
            :W s:triplePattern :T;
79
            :T log:includes { [] example:age :V };
            :W s:triplePattern :U;
80
81
            :U log:includes { :V [] [] }.
82
            };
83
        air:description ("Inclusion includes USE of example:age,
           violated by " :T );
84
        air:assert{:Q air:non-compliant-with :sample-inclusion)};
        # There are more attributes, go to the next policy.
85
86
        air:alt [ air:rule :sample-inclusion-rule-2 ] .
87
88
    :sample-inclusion-rule-2 a air:BeliefRule;
89
        air:label "sample-inclusion example:dob usage check";
90
        air:pattern {
91
        # if RETRIEVE, or alternately, not USE
92
            :P s:var :V;
93
            :W s:triplePattern :T;
94
            :T log:includes { [] example:dob :V }
95
            };
96
        air:description ("Inclusion includes RETRIEVE of example:dob,
           violated by " :T );
97
        air:assert{:Q air:non-compliant-with :sample-inclusion)};
98
        # There are more attributes, go to the next policy.
99
        air:alt [ air:rule :sample-inclusion-rule-3 ] .
100
101
   :sample-inclusion-rule-3 a air:BeliefRule;
102
        air:label "sample-inclusion example:ssn usage check";
103
        air:pattern {
104
        # if USE
105
            :W s:triplePattern :T;
106
            :T log:includes { [] example:ssn :V };
107
            :W s:triplePattern :U;
108
            :U log:includes { :V [] [] }.
109
            };
110
        air:description ("Inclusion includes USE of example:ssn,
           violated by " :T );
111
        air:assert{:Q air:non-compliant-with :sample-inclusion)};
112
        # No more attributes, go to the base case.
113
        air:alt [ air:rule :sample-inclusion-default ] .
114
115
   :sample-inclusion-default a air:BeliefRule;
116
117
        air:label "sample-inclusion default rule";
118
        air:pattern { # Empty search.
119
            };
120
        air:description ("This inclusion policy states that you must do
           all four of these actions: RETRIEVE example:name, USE example
           :age, RETRIEVE example:dob, USE example:ssn. No regulated
           attributes found. Asserting compliance.");
121
        air:assert{ :Q air:compliant-with :sample-inclusion } .
122
123 #ends
```

B.8 Sample Exclusion Policy

```
1
  # Auto-generated AIR exclusion policy.
2
3 @prefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#>.
4 @prefix log: <http://www.w3.org/2000/10/swap/log#>.
5 @prefix math: <http://www.w3.org/2000/10/swap/math#>.
6 @prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#>.
7 Oprefix : <>.
8
9 # User-defined namespaces:
10 Cprefix example: <http://www.example.com/#> .
11
12 @forAll :P, :Q, :T, :U, :V, :W.
13 <code>@forAll</code> :TO, :UO, :VO. <code>#</code> <code>Variables</code> for <code>example:name</code>
14 @forAll :T1, :U1, :V1. # Variables for example:age
15 @forAll :T2, :U2, :V2. # Variables for example:dob
16 @forAll :T3, :U3, :V3. # Variables for example:ssn
17
18 :sample-exclusion a air:Policy;
19
       air:label "A sample auto-generated exclusion policy.";
20
       air:rule :sample-exclusion-check.
21
22 :sample-exclusion-check a air:BeliefRule;
23
       air:label "sample-exclusion query check";
24
       air:pattern {
25
            :Q a s:SPARQLQuery;
26
               s:retrieve :P;
27
               s:clause :W.
28
       };
29
       air:description ("Make sure this is a query.");
       # Goes to the first rule. If there are no attributes,
30
31
       # simply go to the default rule.
32
       air:rule :sample-exclusion-noncompliance-rule.
33
34
   :sample-exclusion-noncompliance-rule a air:BeliefRule;
35
       air:label "sample-exclusion, a default deny policy.";
36
       air:pattern {
37
           # Catch RETRIEVE
38
           :P s:var :VO.
39
           :W s:triplePattern :TO.
40
           :TO log:includes { [] example:name :VO } .
41
           # Catch USE
42
           :W s:triplePattern :T1.
43
           :T1 log:includes { [] example:age :V1 }.
44
           :W s:triplePattern :U1.
45
           :U1 log:includes { :V1 [] [] } .
46
           # Catch RETRIEVE
           :P s:var :V2.
47
48
           :W s:triplePattern :T2.
49
           :T2 log:includes { [] example:dob :V2 } .
50
           # Catch USE
51
           :W s:triplePattern :T3.
```

```
52
           :T3 log:includes { [] example:ssn :V3 }.
53
           :W s:triplePattern :U3.
54
           :U3 log:includes { :V3 [] [] } .
55
           };
       air:description ("You may see up to 3 of 4 of these attributes:
56
           example:name example:age example:dob example:ssn . This query
            is incompliant: " :Q );
       air:assert{ :Q air:non-compliant-with :sample-exclusion } ;
57
58
       # No more policies, go to the base case.
59
       air:alt [ air:rule :sample-exclusion-default ] .
60
61
62
   :sample-exclusion-default a air:BeliefRule;
63
       air:label "sample-exclusion default rule";
64
       air:pattern { # Empty search.
65
           };
66
       air:description ("No regulated attributes found. Asserting
           compliance.");
67
       air:assert{ :Q air:compliant-with :sample-exclusion } .
68
69
  #ends
```

B.9 Sample History-Aware Exclusion Policy

```
# Auto-generated AIR exclusion policy, history aware.
1
2
3 Oprefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#> .
4 @prefix log: <http://www.w3.org/2000/10/swap/log#> .
5 @prefix math: <http://www.w3.org/2000/10/swap/math#>
6 Oprefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#> .
7 Oprefix : <> .
8
9 # User-defined namespaces:
10 Oprefix example: <http://www.example.com/#> .
11
12 @forAll :CURRENT, :H, :HISTORY.
13 <code>@forAll</code> :PO, :QO, :TO, :UO, :VO, :WO. <code>#</code> Variables for <code>example:name</code>
14 <code>@forAll</code> :P1, :Q1, :T1, :U1, :V1, :W1. <code># Variables</code> for <code>example</code>:age
15 @forAll :P2, :Q2, :T2, :U2, :V2, :W2. # Variables for example:dob
16 @forAll :P3, :Q3, :T3, :U3, :V3, :W3. # Variables for example:ssn
17
18
  :sample-exclusion-history a air:Policy;
19
        air:label "A sample auto-generated history aware exclusion
           policy.";
20
        air:rule :sample-exclusion-history-check.
21
22
   :sample-exclusion-history-check a air:BeliefRule;
23
        air:label "sample-exclusion-history query check";
24
        air:pattern {
25
            :CURRENT a s:ComplianceQuery;
26
                      s:Query
                                :Q;
27
                      s:History :HISTORY.
28
       };
```

```
29
       air:description ("Make sure this is a query.");
       # Goes to the first rule. If there are no attributes,
30
31
       # simply go to the default rule.
32
       air:rule :sample-exclusion-history-noncompliance-rule.
33
   :sample-exclusion-history-noncompliance-rule a air:BeliefRule;
34
       air:label "sample-exclusion-history, a default deny policy.";
35
36
       air:pattern {
37
            :HISTORY log:semantics :H .
38
           # Catch RETRIEVE of example:name
39
            :H log:includes {
40
                :QO a s:SPARQLQuery;
41
                    s:retrieve :P0;
42
                    s:clause :WO.
43
                :PO s:var :VO.
44
                :WO s:triplePattern :TO. } .
            :TO log:includes { [] example:name :VO } .
45
46
           # Catch USE of example:age
47
           :H log:includes {
48
                :Q1 a s:SPARQLQuery;
                    s:retrieve :P1;
49
50
                    s:clause :W1.
51
                :W1 s:triplePattern :T1.
52
                :W1 s:triplePattern :U1. } .
53
            :U1 log:includes { :V1 [] [] } .
           :T1 log:includes { [] example:age :V1 }.
54
           # Catch RETRIEVE of example:dob
55
56
            :H log:includes {
57
                :Q2 a s:SPARQLQuery;
58
                    s:retrieve :P2;
59
                    s:clause :W2.
60
                :P2 s:var :V2.
61
                :W2 s:triplePattern :T2. } .
62
           :T2 log:includes { [] example:dob :V2 } .
63
           # Catch USE of example:ssn
           :H log:includes {
64
65
                :Q3 a s:SPARQLQuery;
66
                    s:retrieve :P3;
67
                    s:clause :W3.
68
                :W3 s:triplePattern :T3.
                :W3 s:triplePattern :U3. } .
69
70
           :U3 log:includes { :V3 [] [] } .
71
           :T3 log:includes { [] example:ssn :V3 }.
           };
72
73
       air:description ("You may see up to 3 of 4 of these attributes:
           example:name example:age example:dob example:ssn . The query
          history finds that this new query is incompliant: " :Q );
74
       air:assert{ :Q air:non-compliant-with :sample-exclusion-history
          };
75
       # No more policies, go to the base case.
76
       air:alt [ air:rule :sample-exclusion-history-default ] .
77
78
79
   :sample-exclusion-history-default a air:BeliefRule;
```

```
80 air:label "sample-exclusion-history default rule";
81 air:pattern { # Empty search.
82 };
83 air:description ("No regulated attributes found by this history-
aware exclusion policy. Asserting compliance by default.");
84 air:assert{ :Q air:compliant-with :sample-exclusion-history } .
85
86 #ends
```

B.10 Sample Chaining Policy

```
1 # Auto-generated AIR chaining policy.
2\, # Policy is default non-compliant. Patterns assert compliance.
3
4 Oprefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#>.
5 @prefix log: <http://www.w3.org/2000/10/swap/log#>.
6 Oprefix math: <http://www.w3.org/2000/10/swap/math#>.
7 Oprefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#>.
8 @prefix : <>.
9
10 # User-defined namespaces:
11 Oprefix example: <http://www.example.com/#> .
12
13 @forAll :P, :Q, :T, :U, :V, :W.
14
15 :sample-chaining a air:Policy;
       air:label "A sample auto-generated default-noncompliant chaining
16
           policy.";
17
       air:rule :sample-chaining-check.
18
19
   :sample-chaining-check a air:BeliefRule;
20
       air:label "sample-chaining query check";
21
       air:pattern {
22
           :Q a s:SPARQLQuery;
23
              s:retrieve :P;
24
              s:clause :W.
25
       };
26
       air:description ("Make sure this is a query.");
27
       # Goes to the first rule. If there are no attributes,
28
       # simply go to the default rule.
29
       air:rule :sample-chaining-rule-0.
30
31
   :sample-chaining-rule-0 a air:BeliefRule;
32
       air:label "sample-chaining example:name presence check";
33
       air:pattern {
34
       # if RETRIEVE, or alternately, not USE
35
           :P s:var :V;
36
           :W s:triplePattern :T;
37
           :T log:includes { [] example:name :V } .
38
           };
39
       air:description ("Chaining has found the first pattern RETRIEVE
          in " :T );
40
       air:rule :sample-chaining-rule-1;
```

```
41
       air:alt [ air:rule :sample-chaining-default ] .
42
43
  :sample-chaining-rule-1 a air:BeliefRule;
       air:label "sample-chaining example:age presence check";
44
45
       air:pattern {
       # if USE
46
47
           :W s:triplePattern :T;
            :T log:includes { [] example:age :V };
48
49
           :W s:triplePattern :U;
50
           :U log:includes { :V [] [] }.
51
           };
52
       air:description ("A restriction on the USE of example:age,
           violated by " :T );
53
       air:assert { :Q air:compliant-with :sample-chaining } ;
54
       # There are more policies, go to the next policy.
       air:alt [ air:rule :sample-chaining-rule-2 ] .
55
56
57
   :sample-chaining-rule-2 a air:BeliefRule;
58
       air:label "sample-chaining example:dob presence check";
59
       air:pattern {
60
       # if RETRIEVE, or alternately, not USE
61
            :P s:var :V;
           :W s:triplePattern :T;
62
63
           :T log:includes { [] example:dob :V } .
64
           };
       air:description ("A restriction on the RETRIEVE of example:dob,
65
          violated by " :T );
66
       air:assert { : Q air:compliant-with :sample-chaining } ;
67
       # There are more policies, go to the next policy.
68
       air:alt [ air:rule :sample-chaining-rule-3 ] .
69
70
   :sample-chaining-rule-3 a air:BeliefRule;
71
       air:label "sample-chaining example:ssn presence check";
72
       air:pattern {
73
       # if USE
74
           :W s:triplePattern :T;
75
           :T log:includes { [] example:ssn :V };
76
            :W s:triplePattern :U;
77
            :U log:includes { :V [] [] }.
78
           };
79
       air:description ("A restriction on the USE of example:ssn,
          violated by " :T );
       air:assert { :Q air:compliant-with :sample-chaining } ;
80
81
       # Not done yet! There is at least one filter to check.
82
       air:alt [ air:rule :sample-chaining-filter-0 ] .
83
84
   :sample-chaining-filter-0 a air:BeliefRule;
       air:label "sample-chaining filter check";
85
86
       air:pattern {
            # Test for the filter.
87
88
           :W s:triplePattern :T;
            :T log:includes { [] example:age :V };
89
90
           :V math:lessThan 1991.
91
           };
```

```
92
        air:description ("A filter of the form (example:age math:
           lessThan 1991), matched in " :T );
93
        air:assert { :Q air:compliant-with :sample-chaining } ;
        # There are more filters, go to the next policy.
94
95
        air:alt [ air:rule :sample-chaining-filter-1 ] .
96
97
    :sample-chaining-filter-1 a air:BeliefRule;
98
        air:label "sample-chaining filter check";
99
        air:pattern {
100
             # Test for the filter.
101
            :W s:triplePattern :T;
            :T log:includes { [] example:age :V };
102
103
            :V math:greaterThan 18.
104
            };
105
        air:description ("A filter of the form (example:age math:
           greaterThan 18), matched in " :T );
106
        air:assert { :Q air:compliant-with :sample-chaining } ;
107
        # No more filters, go to the default case.
108
        air:alt [ air:rule :sample-chaining-default ] .
109
110 :sample-chaining-default a air:BeliefRule;
111
        air:label "sample-chaining default rule";
112
        air:pattern { # Empty search.
113
            };
114
        air:description ("No restricted attributes found.");
115
        air:assert { :Q air:non-compliant-with :sample-chaining } ;
116
   #ends
```

B.11 Sample Default Deny Policy

```
1 # Auto-generated AIR default deny policy.
2
3 @prefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#>.
4 Cprefix log: <http://www.w3.org/2000/10/swap/log#>.
5 @prefix math: <http://www.w3.org/2000/10/swap/math#>.
6 @prefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#>.
7 @prefix : <>.
8
9 # User-defined namespaces:
10 Oprefix example: <http://www.example.com/#> .
11
12 CforAll :P, :Q, :T, :V.
13
14
  :sample-defaultdeny a air:Policy;
       air:label "A sample auto-generated default deny policy.";
15
16
       air:rule :sample-defaultdeny-check.
17
   :sample-defaultdeny-check a air:BeliefRule;
18
19
       air:label "sample-defaultdeny query check";
20
       air:pattern {
21
           :Q a s:SPARQLQuery;
22
              s:retrieve :P;
23
              s:clause :W.
```

```
24
       };
25
       air:description ("Make sure this is a query.");
26
       # Goes to the first rule. If there are no attributes,
27
       # simply go to the default rule.
28
       air:rule :sample-defaultdeny-noncompliance-rule.
29
30
   :sample-defaultdeny-noncompliance-rule a air:BeliefRule;
31
       air:label "sample-defaultdeny, a default deny policy.";
32
       air:pattern {
33
           :W s:triplePattern :T;
34
           :T log:notIncludes { [] example:name [] };
35
           :T log:notIncludes { [] example:age [] };
36
           :T log:notIncludes { [] example:dob [] };
37
           :T log:notIncludes { [] example:ssn [] };
38
           };
39
       air:description ("This policy only enables access to example:
          name example:age example:dob example:ssn . Found a triple
          pattern that mentions another predicate: " :T );
40
       air:assert{ :Q air:non-compliant-with :sample-defaultdeny } ;
       # No more policies, go to the base case.
41
42
       air:alt [ air:rule :sample-defaultdeny-default ] .
43
44
45
   :sample-defaultdeny-default a air:BeliefRule;
46
       air:label "sample-defaultdeny default rule";
47
       air:pattern { # Empty search.
48
           };
49
       air:description ("We could not find any attributes that must be
           included. Asserting compliance by default.");
50
       air:assert{ :Q air:compliant-with :sample-defaultdeny } .
51
52 #ends
```

B.12 No-Address Restriction Policy for Sample Scenario

```
1 # Auto-generated AIR restriction policy.
2
3 Cprefix air: <http://dig.csail.mit.edu/TAMI/2007/amord/air#>.
4 @prefix log: <http://www.w3.org/2000/10/swap/log#>.
5 @prefix math: <http://www.w3.org/2000/10/swap/math#>.
6 Oprefix s: <http://dig.csail.mit.edu/2009/IARPA-PIR/sparql#>.
7 @prefix : <#>.
8
9 # User-defined namespaces:
  @prefix ex: <http://example.com/#>.
10
11
12
13 CforAll :P, :Q, :T, :U, :V, :W.
14
15 :no-address a air:Policy;
```

```
16
       air:label "Users may not find the home address of members of the
            database.";
17
       air:rule :no-address-check.
18
19
   :no-address-check a air:BeliefRule;
       air:label "no-address query check";
20
21
       air:pattern {
22
            :Q a s:SPARQLQuery;
23
               s:retrieve :P;
24
              s:clause :W.
25
       };
26
       air:description ("Make sure this is a query.");
27
       # Goes to the first rule. If there are no attributes,
28
       # simply go to the default rule.
29
       air:rule :no-address-rule-0.
30
31
   :no-address-rule-0 a air:BeliefRule;
32
       air:label "no-address ex:address restriction";
33
       air:pattern {
       # if RETRIEVE, or alternately, not USE
34
           :P s:var :V.
35
36
            :W s:triplePattern :T.
37
           :T log:includes { [] ex:address :V }. #2
38
           };
39
       air:description ("A restriction on the RETRIEVE of ex:address,
          violated by " :T );
       air:assert{:Q air:non-compliant-with :no-address};
40
       # There are more policies, go to the next policy.
41
42
       air:alt [ air:rule :no-address-rule-1 ] .
43
   :no-address-rule-1 a air:BeliefRule;
44
45
       air:label "no-address ex:address restriction";
46
       air:pattern {
       # if USE
47
48
           :W s:triplePattern :T.
           :T log:includes { [] ex:address :V }. #1
49
            :W s:triplePattern :U.
50
51
           :U log:includes { :V [] [] }.
52
           };
53
       air:description ("A restriction on the USE of ex:address,
           violated by " :T );
       air:assert{:Q air:non-compliant-with :no-address};
54
55
       # No more policies, go to the base case.
56
            air:alt [ air:rule :no-address-default ] .
57
58
   :no-address-default a air:BeliefRule;
59
       air:label "no-address default rule";
60
       air:pattern { # Empty search.
61
            };
62
       air:description ("No restricted attributes found.");
       air:assert{ :Q air:compliant-with :no-address }.
63
```

Bibliography

- L. Kagal, "Policy assurance for pir queries." http://dig.csail.mit.edu/2009/ Talks/0226-IARPA-PI/policy-assurance.pdf, 2009.
- [2] S. Bratt, "Semantic web, and other technologies to watch." http://www.w3. org/2007/Talks/0130-sb-W3CTechSemWeb/, 2007.
- [3] I. Herman, R. Swick, and D. Brickley, "Resource description framework (rdf)." http://www.w3.org/RDF/, 2004.
- [4] E. Prud'hommeaux and A. Seaborne, "Sparql query language for rdf." http: //www.w3.org/TR/rdf-sparql-query/, January 2008.
- [5] MySQL A.B. and Sun Microsystems, "Mysql 6.0 reference manual." http:// dev.mysql.com/doc/refman/6.0/en/index.html, 2009.
- [6] R. K. Thomas and R. S. Sandhu, "Discretionary access control in object-oriented databases: Issues and research directions," in *National Computer Security Conference*, pp. 63–74, September 1993.
- [7] L. Qiu, Y. Zhang, F. Wang, M. Kyung, and H. R. Mahajan, "Trusted computer system evaluation criteria," in *National Computer Security Center*, 1985.
- [8] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," in 15th National Computer Security Conference, pp. 554–563, October 1992.
- [9] H. Li, X. Zhang, H. Wu, and Y. Qu, "Design and application of rule based access control policies," in *ISWC Workshop on Semantic Web and Policy*, pp. 34–41, 2005.
- [10] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman, "Information accountability," *Communications of the ACM*, June 2008.
- [11] E. Prud'hommeaux, "Swobjects source code directory." http://www.w3.org/ 2008/04/SPARQLfed/, 2008.
- [12] E. Prud'hommeaux, "Spasql: Sparql support in mysql." http://www.w3.org/ 2005/05/22-SPARQL-MySQL/XTech, June 2007.

- [13] L. Feigenbaum, "Dawg testcases." http://www.w3.org/2001/sw/DataAccess/ tests/r2, 2004.
- [14] L. Kagal, T. Berners-Lee, D. Connolly, and D. Weitzner, "Using semantinc web technologies for policy management on the web," in 21st National Conference on Artificial Intelligence (AAAI), July 2006.
- [15] O. Seneviratne, L. Kagal, D. Weitzner, H. Abelson, T. Berners-Lee, and N. Shadbolt, "Detecting creative commons license violations on images on the world wide web," in WWW2009, April 2009.
- [16] O. W. Seneviratne, "Framework for Policy Aware Reuse of Content on the WWW," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2009.
- [17] J.-W. Byun, "Access control features in oracle." http://www.cs.purdue.edu/ homes/ninghui/courses/Spring05/lectures/Oracle.pdf, April 2005.
- [18] L. P. Hale and J. Levinger, "Oracle database security guide, 10g release 1." http://www.stanford.edu/dept/itss/docs/oracle/10g/network. 101/b10773/title.htm, December 2003.
- [19] R. Ramaswamy and R. S, "Role-based access control features in commercial database management systems," in *In Proceedings of 21st NIST-NCSC National Information Systems Security Conference*, pp. 503–511, 1998.
- [20] C. Y. Chung, M. Gertz, and K. Levitt, "Demids: A misuse detection system for database systems," in *In Third International IFIP TC-11 WG11.5 Working Conference on Integrity and Internal Control in Information Systems*, pp. 159– 178, Kluwer Academic Publishers, 1999.
- [21] R. Cathey, L. Ma, N. Goharian, and D. Grossman, "Misuse detection for information retrieval systems," in ACM 12th Conference on Information and Knowledge Management, 2003.
- [22] A. Kamra, E. Terzi, and E. Bertino, "Detecting anomalous access patterns in relational databases," VLDB Journal: The International Journal on Very Large Data Bases, 2007.
- [23] L. Sweeney, "k-anonymity: A model for protecting privacy," International Journal on Uncertainty, Fuzziness, and Knowledge-Based Systems, vol. 10 (5), pp. 555–570, 2002.
- [24] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *International Journal on Uncertainty, Fuzziness, and Knowledge-Based Systems*, vol. 10 (5), pp. 571–588, 2002.
- [25] L. Sweeney, "Guaranteeing anonymity when sharing medical data: the datafly system," in AMIA Annual Fall Symposium, 1997.

- [26] A. D. Mascio, N. Chauvat, and S. Thenault, "fyzz is a sparkling python parser for the sparql query language." http://www.logilab.org/project/fyzz, 2009.
- [27] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," Scientific American, May 2001.
- [28] H. Halpin, S. Hawke, I. Herman, E. Prudhommeaux, D. Raggett, and R. Swick, "W3c semantic web activity." http://www.w3.org/2001/sw/, 2009.
- [29] Adobe, "Adobe xmp partners." http://www.adobe.com/products/xmp/ partners.html, 2009.
- [30] C. Commons, "Describing copyright in rdf." http://creativecommons.org/ns, 2009.
- [31] T. Berners-Lee and D. Connolly, "Naming and addressing: Uris, urls," http: //www.w3.org/Addressing/, 1997.
- [32] D. Beckett and B. McBride, "Rdf/xml syntax specification." http://www.w3. org/TR/rdf-syntax-grammar/, 2004.
- [33] D. L. McGuinness and F. van Harmelen, "Owl web ontology language overview." http://www.w3.org/TR/owl-features/, 2004.
- [34] T. Berners-Lee, "Tabulator: Generic data browser." http://www.w3.org/2005/ ajar/tab, 2005.
- [35] J. Rusher, "Rhetorical device: Triple store." http://www.w3.org/2001/sw/ Europe/events/20031113-storage/positions/rusher.html, 2001.
- [36] C. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," Artificial Intelligence, vol. 19, p. 1737, 1982.
- [37] T. Berners-Lee, "Semantic web application platform swap." http://www.w3. org/2000/10/swap/, 2005.
- [38] T. Berners-Lee, "cwm." http://www.w3.org/2000/10/swap/doc/cwm.html, 2008.
- [39] G. J. Sussman, G. L. S. Jr., C. Rich, J. Doyle, and J. de Kleer, "Amord: A deductive procedure system," *MIT Artificial Intelligence Laboratory*, 1977-08.
- [40] L. Kagal, "Scenario 0: Mit prox card policy violation." http://dig.csail.mit. edu/TAMI/2007/s0/, 2007.
- [41] L. Kagal, "Air policy tutorial." http://tw.rpi.edu/proj/tami/AIR_Policy_ Tutorial, 2009.