INTELLIGENT INFORMATION GATHERING FOR OPEN WORLD POLICY REASONING

A Thesis

by

İ. Can Büyükyıldız

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Computer Science

Özyeğin University
August 2015

INTELLIGENT INFORMATION GATHERING FOR OPEN WORLD POLICY REASONING

Approved by:

Assoc. Prof. Murat Şensoy , Advisor
Department of Computer Science
Özyeğin University

Assist. Prof. Cenk Demiroğlu
Department of Computer Science
Özyeğin University

Assoc. Prof. Pınar Yolum
Department of Computer Engineering
Boğaziçi University

Date Approved: 3 August 2015

ABSTRACT

Policies play an important role in autonomous multi-agents systems where each agent aims to achieve its own goals. Policies and related mechanisms allow authority or society to regulate the actions of agents to prohibit malicious and undesirable activities. Without policies, society could be harmed by irresponsible and malicious activities of its members. On the other hand, reasoning with policies is not trivial; it requires extensive knowledge about the environment. If the knowledge is incomplete or missing, reasoning with policies may not be possible. In this thesis, we propose a proactive approach for gathering information to reason with policies. While our approach can be used in various settings, we provide two case-studies; one in social networking domain and the other in on-line advertisement domain. Through experiments we demonstrated that our approach allows high rate of success during policy reasoning when the knowledge bases is not complete.

ÖZETÇE

Sistem politikaları her birimin kendi hedeflerine ulaşmaya çalıştığı çoklu etmen sistemlerinde önemli bir rol oynamaktadır. Politikalar ve ilgili mekanizmalar birimlerin kötü niyetli ve istenmeyen faaliyetlerini engellemek için uygun yetki ya da ortamı sağlarlar. Politikalar olmadan, üyelerinin sorumsuz ve kötü niyetli faaliyetlerinden sistemler zarar görebilir. Politikaların uygulanabilmesi için ortam hakkında geniş bilgiye sahip olunması gerekir. Eğer ortam hakkında bilgi yeterli değil ise, politikaların başarılı bir şekilde uygulanması mümkün olmayabilir. Bu tezde politikalarin başarıyla uygulanabilmesi için bilgi toplamaya yönelik proaktif bir yaklaşım öneriyoruz. Bu yaklaşımın uygulanabilirliğini göstermek için birisi sosyal ağ alanında, diğeri ise çevirimiçi reklam alanında olmak üzere iki örnek ele alıyoruz. Yaptığımız testler ile yaklaşımımızın bilgi tabanının eksik olduğu durumlarda politika uygulanabilirliğini arttırmada yüksek oranda başarı sağladığını gösteriyoruz.

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Dr. Murat Şensoy. Without his continuous advice, guidance and support this dissertation would not have been possible. I am also profoundly thankful to my family and my wife for their patience and encouragement during my study and all through my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CWA	Closed World Assumption
OWA	Open World Assumption
IoT	Internet of Things
PCIM	Policy Core Information Model
IETF	Internet Engineering Task Force
DMTF	Distributed Management Task Force
PR	Policy Repository
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PDL	Policy Description Language
XACML	eXtensible Access Control Markup Language
DL	Description Logic
OWL	Web Ontology Language
FOL	First-Order Logic
PRISM	PRogramming In Statistical Modeling
ILP	Inductive Logic Programming
HAIL	Hybrid Abductive Inductive Learning
SOA	Service-Oriented Architecture
UDDI	Universal Description, Discovery and Integration
API	Application Programming Interface
WSDL	Web Services Description Language
XML	Extensible Markup Language
HTTP	Hyper-Text Transfer Protocol
SMTP	Simple Mail Transfer Protocol
WAS	WebSphere Application Server
WMB	WebSphere Message Broker
IG	Information Gatherer
SD	Service Description

OCR	Optical Character Recognition
POC	Proof Of Concept

CHAPTER I

INTRODUCTION

Policies are an important part of human societies. Legislations, traffic regulations, health and safety rules are all examples of policies we have. Policies regulate our actions through *permissions*, *prohibitions*, and *obligations*. For instance, the following simple policies in health and safety domain may be life-saving: i) people are prohibited to enter a coal mine if the oxygen level is low, ii) if methane level is normal or low, miners are permitted to enter to a mine, iii) if methane level is high, everyone in a mine are obliged to leave the mine. On the other hand, without information about gas levels in a mine, these policies are useless.

Policy makers create policies to regulate systems. However, policies could not be in effect without the necessary information. For instance, health and safety policies above require measurements of oxygen and methane levels in coal mines. If the knowledge base is missing necessary information, either policy reasoning would be prematurely failed or simply policies would not be effective.

Most of the policy reasoning frameworks are based on Logic Programming and adopt Closed World Assumption (CWA). In these frameworks, when something is not known, it is assumed *false*. This is called negation by failure. For instance, if the oxygen level is not known, every proposition related to oxygen levels are assumed *false*. Therefore, people do not have to leave a coal mine if the oxygen level in the mine is unknown; this is simple because of the fact that the preconditions of policies related to oxygen level fail. It is not surprising that CWA misleads policy reasoning by using negation by failure.

There are other policy reasoning approaches based on Open World Assumption

(OWA). Unlike CWA, reasoning with OWA does not assume that unknown information is false; instead it considers it as unknown. Therefore, the truth value of a boolean expression can be *true*, *false*, or *unknown.* For instance, in our previous example, if the oxygen level is not known, every proposition related to oxygen levels are assumed *unknown.* Therefore, we cannot reason with the policies related to oxygen level; these policies could not be in effect due to missing information.

In this paper, we propose a novel approach to intelligently gather missing information for policy reasoning. Our approach uses *Abductive Reasoning* to determine missing information to reason with a specific policy. Then, we determine which information services should be contacted to gather missing information. For this purpose, we use HyperCat [1], which is a state-of-the-art, open, lightweight JSON-based hypermedia catalogue format developed specifically for Internet of Things (IoT). There may be more than one service providing the same information with different cost and precision. Therefore, our approach selects the best information services within a given budget to increase precision.

CHAPTER II

RELATED WORK

In this section, we overview the existing work related to our research presented in this paper.

2.1 Policy Representation

There are several policy frameworks proposed in the literature and implemented for practical use. A significant portion of them are based on PCIM (Policy Core Information Model) developed jointly by IETF and DMTF. It provides the policy-driven management consists of the following components: PR (Policy Repository), PDP (Policy Decision Point), and PEP (Policy Enforcement Point). In these frameworks, administrators define and edit policies in the form of IF (Condition) THEN (Action) rules, where conditions and actions are described using propositions and boolean expressions. Existing policy representations languages such as the Policy Description Language (PDL) of Bell labs and the OASIS standard XACML (eXtensible Access Control Markup Language) are based on rules in this form. Many commercial policy frameworks such as IBM Tivoli and HP Openview PolicyXpert are using XACML to represent policies. In this work, we also assume that policies are represented using IF (Condition) THEN (Action) rules. This allows us to focus on our main contribution – information gathering for policy reasoning. Although most of the commercial policy frameworks are based on simple if-then rules, more expressive policy languages are proposed in the literature.

Ponder [2] is a declarative and object-oriented policy language from Imperial College. It uses a propositional logic programming to declare role-based access control policies. There are five types of policies in Ponder: Authorization policies, Filter

Policies, Refrain policies, Delegation policies, and Obligation policies. Ponder2 [3] is an extension of Ponder with domain service that provides a hierarchical structure for managing objects.

Rei [4] is a policy language based on a subset of Web Ontology Language (OWL-Lite) and Prolog. It allows logic-like variables to be used while describing policies. This gives it the flexibility to specify relations like *role value maps* that are not directly possible in OWL. The use of these variables, however, makes Description Logics (DLs) reasoning services (e.g., static conflict detection between policies) unavailable for Rei policies.

KAoS [5] is, probably, the most developed language for describing policies that are built upon Web Ontology Language (OWL). KAoS was originally designed to use OWL-DL to define actions and policies. This, however, restricts the expressive power to DL and prevents KAoS from defining policies in which one element of an action's context depends on the value of another part of the current context. KAoS distinguishes between (positive and negative) obligation policies and (positive and negative) authorization policies. Authorization policies permit (positive) or forbid (negative) actions, whereas obligation policies require (positive) or do not require (negative) action. Actions are also the object of a KAoS policy, and conditions on the application of policies can be described (context), although the subject (individual/role) of the policy is not explicit (it is, however, in Rei).

OWA and CWA lead to two different approaches in evaluating implicit knowledge. In the open-world assumption, we can not assume some information is false because it does not exist in our knowledge base. In CWA, the main assumption is that the unknown knowledge is false, which is called *negation by failure*. Unlike the CWA, OWA assumes knowledge base is incomplete. The following example demonstrate the fundamental difference between these two approaches:

- Premise - "John is watching TV"

- Premise - ”Mary is watching TV”
- Question - ”Is Bill watching TV?”
- **CWA** answer - No.
- **OWA** answer - Unknown.

In this work, we argue that considering unknown data as false may damage the governance of system. Therefore, our approach gathers missing information for a more complete policy reasoning. To clarify our approach, we represent our policies in the form of Horn clauses, which are similar to Prolog clauses.

Logic programming languages, such as Prolog, are based on CWA and uses negation by failure. Most of the existing policy languages are based on logic programming, therefore they use negation by failure during policy reasoning. Ponder and Rei are examples of such policy languages. On the other hand, DLs is a decidable fragment of First-Order Logic (FOL) and based on OWA. Policy languages such as KAoS and OWL-Polar are using OWL, which is underpinned by DLs. If precondition for a policy contains some unknown predicates, a policy reasoner using OWA cannot conclude that whether the policy is activated or not. In practice, the outcome would be the same, the policy would not get activated due to lack of knowledge.

Policies are important to protect users' privacy in online social networks. PeopleFinder, a location sharing platform, is an example for allowing its users to define access policies. Users can select when and who can see their shared location [6]. Sadeh *et al.*. indicates in the research that more research should be conducted in order to understand users' need when they are specifying their access preferences.

In dynamic systems like social networks, information come in to play in different forms. Policies depend on the static nature of the content but the information needs to be protected is not necessarily represented in its own form; it can be inferred from another information. Klemperer et al. have researched a new approach for access controls for photo sharing [7]. They used tags, that are usually used for organisation, to define access control policies.

2.2 Abductive Reasoning

Abduction is the process of finding statements that should be added to a knowledge base to entail a specific conclusion. It has applications especially in diagnosis domain. For instance, a doctor can reason about a disease based on the symptoms of a patient using abductive reasoning. D. Poole presents a framework [8] that uses Horn-clause abduction and assigns probability value to hypotheses. It provides a combination between logical and probabilistic reasoning and provides an evidence to be used for abduction and assumption-based reasoning.

PRISM (PRogramming In Statistical Modeling) [9] is language based on Prolog that subsumes several statistical tools to support probability theory and combines it with learning. The system uses abduction to find distributions of explanations for facts. It consists of three stages, namely sample execution, probability calculation and learning. An approach [10] uses PRISM to implement an efficient system toward a statistical modelling. They make use of PRISM's naive learning algorithm and improve it in the means of explanations and compilation to support real-world applications.

The integration of abduction and induction is widely investigated concept in decision making process. Peter Flach *et al.* [11] introduced a knowledge development framework that uses abduction and induction integration. They use a cycle of integration that uses abducible predicates – set of predicates that are allowed to appear in hypotheses– and abduction to transform observations to informations and use these informations as input to induction. Induction tries to create observable predicates from abducible predicates and with this learned information cycle repeats.

Another approach *Abductive Concept Learning* [12] integrates abduction and induction to create a learning framework. It extends Inductive Logic Programming by considering background and target theories as abductive theories. The framework also provides learning with incomplete knowledge base by exploiting abduction's hypothetical reasoning.

O. Ray identifies incompleteness of ILP (Progol System [13]) proof procedure and proposes a new approach called Hybrid Abductive Inductive Learning (HAIL) [14] that integrates abduction and induction within a learning cycle. It overcomes the incompleteness by computing multiple clauses in response to single seed example and by finding explanations that can not derived by Bottom Generalisation [15].

DAREC [29] is a distributed multi-agent abductive policy reasoning framework with arithmetic constraint support. It allows collaborative abductive reasoning between decentralised agents. The information (assumptions and constraints) is shared between agents and checked for global consistency before policy reasoning.

2.3 Information Services

In order to gather missing information for policy reasoning, we may discover and query information services using a Service-Oriented Architecture (SOA) perspective. For this purpose, we use a dynamic registry service that stores meta-information about services and serves as a registry at run time.

UDDI (Universal Description, Discovery and Integration) [16] is a standardised directory that provides listing for applications to describe their services and methods (APIs) required to work with. UDDI provides interoperable, foundational infrastructure using common industry standards, such as XML, XML based Web Services Description Language (WSDL), and SOAP. UDDI utilises SOAP specification for collaboration between web services. SOAP[1] uses application layer protocols (e.g HTTP,

[1]http://www.w3.org/TR/soap

SMTP) to provide exchanging structured data between applications. UDDI uses WSDL interface for web services to describe their functionality and protocols in order to interact with them.

Another service catalogue is WSO2 Governance Registry[2] which provides SOA integrated registry database that stores content repository and governance framework. It's a complete tool for a SOA platform as it provides a framework that you can manage contents/services. WSO2 Governance Registry stores service information using WSDL, XML and WS-Policy. WebSphere Application Server (WAS)[3] and WebSphere Message Broker (WMB) provide a similar services that can be used for SOA environment [17]. WAS is actually a framework that hosts Java based web applications. WMB may be used for application connectivity and data exchange between applications.

The service registries above are mostly XML based frameworks. XML is well formed markup language extensively used to store and transport data. However, web services are migrating to the JSON based APIs rather then using XML based APIs. As an example, Twitter has stopped support on XML based streaming API in 2010 and migrated to JSON. JSON is considerably easy to be read by human and easy to generate and parse by computers[4]. There are many case studies regarding performance differences of XML and JSON [18]. As we implement our framework in JS and JSON provides significant performance boost, we will use a registry service that is based in JSON.

HyperCat [1] is a JSON-based RESTful hypermedia catalogue server mainly developed to solve Internet of Things (IoT) interoperability. A catalogue in HyperCat server is basically an array of URIs each annotated with metadata. Services can register to catalogues and indicate what data they can provide. Most importantly,

[2]https://docs.wso2.com/display/Governance452
[3]http://www.ibm.com/software/websphere
[4]http://json.org/

Table 1: Metadata from HyperCat Specification

rel	*meaning*	*value*
urn:X-tsbiot:rels:hasDescription:en	Resource has a human readable description in English. Mandatory.	string
urn:X-tsbiot:rels:isContentType	Data provided by resource is of given type. Mandatory if resource is a catalogue.	"application/ vnd.tsbiot.catalogue+json" for catalogues, else RFC2046 MIME type as JSON string (e.g., "text/csv")
urn:X-tsbiot:rels:hasHomepage	A reference to a human readable web page concerning the resource.	URL as a JSON string
urn:X-tsbiot:rels:containsContentType	The catalogue contains resources of given content type. Only meaningful for metadata objects contained by or pointing to catalogue objects.	RFC2046 MIME type as JSON string (e.g., "text/csv")
urn:X-tsbiot:rels:supportsSearch	This catalogue supports a search mechanism. Only meaningful for metadata objects contained by or pointing to catalogue objects.	"urn:X-tsbiot:search:simple" if supports simple search

there is no limitation in defining metadata properties and there is no fixed categories and labels. Developers may choose and create any property that would suit their needs. So that, services can indicate any information to let other services know. However, to empower interoperability, few commonly used metadata properties are defined in the HyperCat specification[5]. Table 1 represents the metadata defined in this specification.

Pathfinder[6] is a simple HyperCat catalogue server for IoT resources and clients. It supports read, create, modify and delete operations in catalogues with basic authentication. Pathfinder doesn't provide or host any user applications, it only provides a registry hub for user applications to share their APIs for other clients. We use Pathfinder as our database for external services to register and share their APIs with certain metadata specifications.

[5]http://wiki.1248.io/doku.php?id=hypercat
[6]http://wiki.1248.io/doku.php?id=pathfinder

CHAPTER III

SYSTEM OVERVIEW

We show an overview of our framework in Figure 1. It is composed of five main components. Policy store is used as a database for existing policies in the system. *Knowledge Base* represents the information known at the moment. *Policy Reasoner* uses knowledge base and policy store to reason with policies. At the heart of policy reasoner, we have *Abductive Reasoner* that detects missing information necessary for policy reasoning. Once missing information is determined, it is reported to *Information Gatherer* (IG), which contacts a service registry – *HyperCat Server* – to get meta-data about available services that provide missing information.

Meta-data related to a service contains which information the service provides, its cost, and precision advertised by the service. Based on this, IG decides which services to query to get information. For instance, in order to learn if the proposition *methanLevel(mine3,high)* –methane level is high in *mine*3– is *true*, it may query a gas sensor in the mine. The answer of this query may be a single probability value for the proposition's truthfulness or a beta distribution that indicates the likelihood of each such probabilities [19]. If the precision of the sensor is low, IG may query more than one sensor and fuse their answers. Lastly, IG updates the knowledge base with the gathered information.

Selection of information services to query may based on their precision. However, the advertised precisions may not be correct. Initially, IG uses the advertised precisions, but it models the actual precision of these services over time using statistical methods in order to improve service selection.

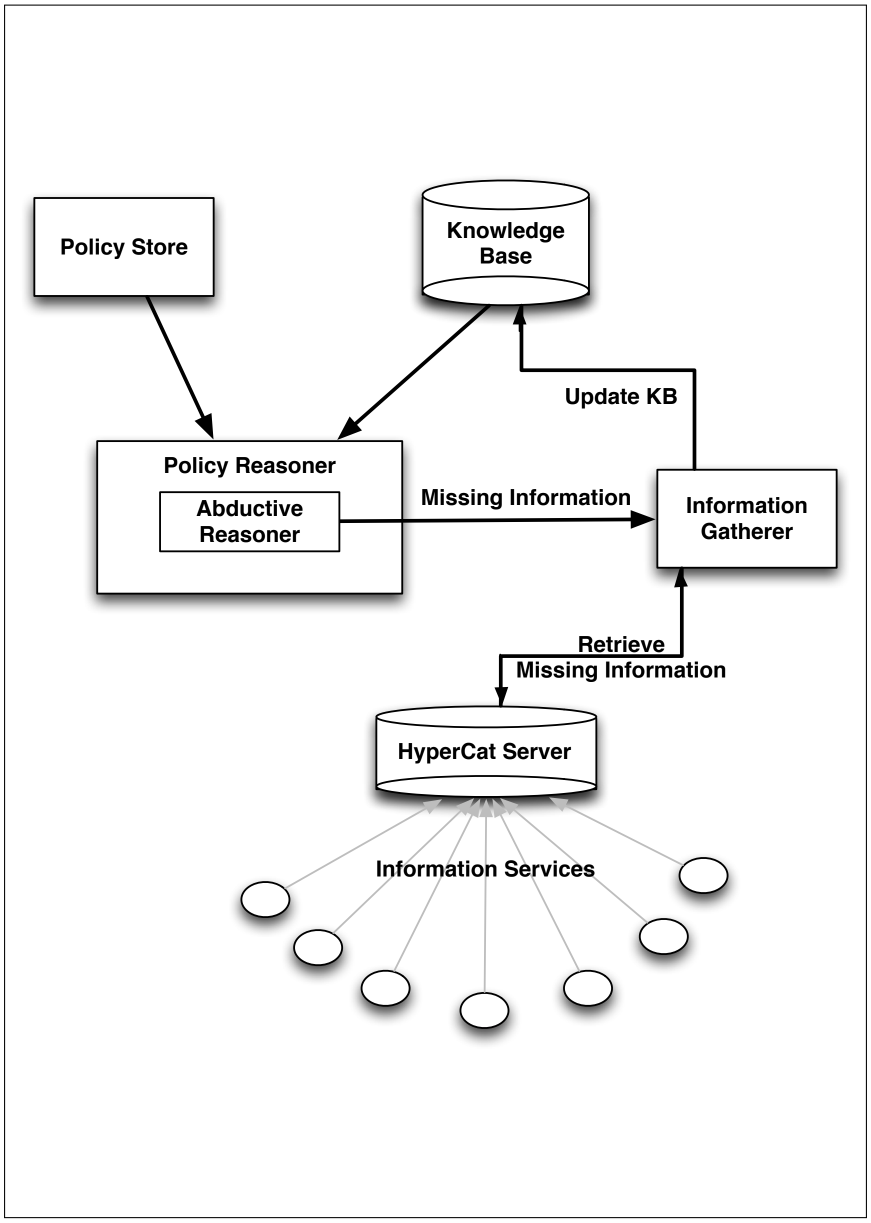


Figure 1: System architecture.

CHAPTER IV

POLICY REPRESENTATION

We represent policies as a set of simple horn clauses. Horn clauses are the basis of logic programming, where clauses are defined in the form of implication: $(p \wedge q \wedge \cdots \wedge t) \rightarrow u$. Therefore, in this work, we use Prolog to define policies. Prolog is a declarative logic programming language which consists of relations (clauses) for program logic and queries for execution. It is created based on procedural interpretation of Horn clauses and extended by addition of negation as failure. A clause in Prolog is defined in the form of:

```
Head :- Body
```

Head and Body in clauses are defined using *terms*. A term could be atoms, numbers, variables or compound terms.

Atom is a name used to refer a static object. Atom must start with lower-case letter.

E.g., *john, x, atom.*

Number is an integer or float numeral.

E.g., *2015, 1, 5*

Variable starts with an upper-case letter or an underscore.

E.g., *Var, _var2, X*

Compound term is composed of an atom and a set of terms in parenthesis, separated by commas.

E.g., *birth_day(john,1945), friends(marry,sally)*

If a clause is defined with a **Body**, it represents a **Rule** and a **Head** is true only if **Body** is true. If **Body** is not defined, it denotes a **Fact**. Facts can be represented as follows:

```
human(john).
coalMine(mine3).
in(tunnel2, mine3).
in(john, tunnel2).
methanLevel(mine3, high).
```

We can also define some rules like the ones below:

```
in(A,C) :- in(A,B), in(B,C).
mustLeave(P,M) :- human(P),
                  coalMine(M),
                  in(P, M),
                  methanLevel(M,high).
```

We formalize policies using such rules. For instance, the rule above represents the policy: "a human in a mine with high methane level must leave the mine". If we query Prolog engine with *mustLeave(P,M)*, it returns *mustLeave(john,,mine3)* – john must leave the mine.

In our implementation, we use tuProlog [20], which is a Java-based Prolog interpreter designed around a minimal core and it can be dynamically configured by loading/unloading libraries. The tuProlog engine is also designed to be exploited straight from Java. We use tuProlog's ability to support multi-paradigm programming between Java and Prolog to assert *facts* and *rules* in order to populate and update our knowledge base and do policy reasoning in runtime.

CHAPTER V

ABDUCTIVE REASONER

Abduction is an important reasoning service which provides possible explanations (or hypotheses) for observations that are not entailed by the current knowledge. In this section, we briefly formalize the notion of abduction.

An abduction problem in our work is a tuple $(\mathcal{K}, \mathcal{H}, A)$, where $\mathcal{K}$ is a knowledge base, called the background knowledge; $\mathcal{H}$ is a set of predicates, which are called abducibles; and A is a grounded predicate such that $\mathcal{K}$ does not entail A, i.e., $\mathcal{K} \not\models A$. A solution to an abduction problem $\mathcal{P} = (\mathcal{K}, \mathcal{H}, A)$ is a set $\mathcal{S} = \{C(t_1, \ldots, t_n) \mid C \in \mathcal{H}\}$ of assertions such that:

1. Each t_i is a ground term, e.g., literal, object etc.
2. The knowledge base $(\mathcal{K} \cup \mathcal{S})$ is consistent,
3. $(\mathcal{K} \cup \mathcal{S}) \models A$

The most state-of-the-art abduction systems [21, 22] are built on Prolog engines that work on plain datalog programs. Du *et al.* described a procedure to translate a datalog program into a Prolog program. That is, using a chain of transformation, we can convert datalog knowledge base into a Prolog program $\mathcal{K}$. Then, we can solve the abductive reasoning problem using $\mathcal{K}$ and existing abductive reasoning methods for plain datalog programs [23].

Figure 2 shows a simplified Prolog program for abductive reasoning over $\mathcal{K}$. This simple program is composed of six rules. Using only six rules, this program defines the predicate *abduce*$(A, \mathcal{S})$, where A is an axiom such as *mustLeave(P,M)* and $\mathcal{S}$ is a solution to the abduction problem (i.e., abductive explanation) computed by

```
1. abduce(A,S):-
      abduce(A,[],S).
2. abduce(A,S,S):-
      holds(A),!.
3. abduce((A,B),S0,S):-!,
      abduce(A,S0,S1),
      abduce(B,S1,S).
4. abduce(A,S0,S):-!,
      clause(A,B),
      abduce(B,S0,S).
5. abduce(A,S,S) :-
      member(A,S),!.
6. abduce(A,S,[A|S]):-
      abducible(A),
      checkConsistency([A|S]).
```

Figure 2: Simplified abductive reasoner for Prolog.

the program. For this purpose, it simply starts with an empty set of axioms as shown in the rule 1 and populates it iteratively with the necessary assertions based on other rules. The rule 2 guarantees that already entailed ABox axioms do not appear in $\mathcal{S}$. The rule 4 expands a complex ABox axiom into its components by finding a clause in $\mathcal{K}$ so that the head of the clause unifies the axiom. The rule 5 prevents redundancies in the solution. The rule 6 expands an existing partial solution by adding a new axiom if this axiom is an *abducible* and this addition does not create an inconsistency. Abducibles correspond to $\mathcal{H}$, i.e., the predicates that we desire to appear in the solution. If a predicate does not appear in the head of any clause in the Prolog knowledge base, then it should also be an abducible. We may note that clauses in $\mathcal{K}$ may results in cycles. For instance, a Prolog clause "$human(X) :- hasParent(X,Y), human(Y)$" may lead to cycles. For the sake of simplicity, we have not shown it in Figure 2, but we implemented rule 4 so that it does not expand an axiom if this expansion results in a loop, instead this axiom is added directly to the solution. In this way, we prevent infinite loops during abductive reasoning.

Let us show how abduction works through the previous simple example. In our example, we have the following Prolog knowledge base.

```
human(john).
coalMine(mine3).
in(tunnel2, mine3).
in(john, tunnel2).
in(A,C) :- in(A,B), in(B,C).
mustLeave(P,M) :- human(P),
               coalMine(M),
               in(P, M),
               methanLevel(M,high).
methanLevel(M,high) :- hasMethanCon(M,X),
                       X > 0.05.
```

That is, we know that John is human in a tunnel, which is in a mine. We also have a policy forcing people to leave a mine if its methane level is high. When we run abduction with using the predicate $abduce(mustLeave(john, mine3), \mathcal{S})$, the algorithm returns $\mathcal{S} = \{hasMethanCon(mine3, X)\}$. This means that we need to know the methane concentration in air to determine whether the mine has high level of methane or not. After having information, we can decide if John must leave the mine.

As a result of abduction, we get a list of unknown assertions that are necessary to reason with policies, e.g., *methanLevel(mine3,high)*. The next step would be locating information services that may provide the unknown information. In the next section, we describe in detail how we use a service directory for this purpose.

CHAPTER VI

SERVICE DIRECTORY

In this work, we used Pathfinder – HyperCat based catalogue server – as a service directory. In HyperCat, metadata is used to describe services. Possible metadata properties are listed in Table 1. Besides these mandatory properties, we also defined custom metadata properties to represent service specific data that will be used in service selection and information gathering. See Table 2.

Catalogue operations are handled with simple HTTP requests. To read a catalogue in JSON format, GET request on the catalogue URL should be used. To create, insert or update a catalogue item, JSON item object should be send to catalogue URL with a POST request. To delete an item, a DELETE command should be requested with a query parameter of "href" value of the corresponding item. Table 4 shows status codes returned from server.

A catalogue is a JSON object that contains an array of metadata objects and list of items which may also represent a catalogue. The "urn:X-tsbiot:rels:hasDescription:en" and "urn:X-tsbiot:rels:isContentType" mandatory properties are used to define our catalogue. After the creation of the catalogue, external services may register. We implemented our own sample services to be able to demonstrate our framework as proof of concept. These services are used to gather information to complete our knowledge base. We register these services to the catalogues. A service description SD is an 8-tuple $\{L, D, R, T, I, V, C, P\}$, where L is the location of the information service, D is the human readable service description, R is the format of the data provided by information service, T is the type of information service it provides, I is the type of the data that information service gets as input, V is the name of the variable that

Table 2: Custom Metadata for Service Details

rel	*meaning*	*value*
urn:X-tsbiot:rels:informationType	Data supported by resource is of given type	See Table 3
urn:X-tsbiot:rels:inputType	Hypertext transfer protocol supported by service	RFC1945 type as string (e.g., GET)
urn:X-tsbiot:rels:inputVariable	Data provided to service is of given type	We used URLs for inputs in both our examples
urn:X-tsbiot:rels:precision	Success rate of service	Rate in form of number out of 10
urn:X-tsbiot:rels:cost	Value of service's cost	A number between 0 to 10. 10 being the most expensive

Table 3: Possible Information Types for Services as used in each Example

Example	*type*1	*type*2	*type*3
Example 1	location	people	
Example 2	keyword	reference	link

Table 4: Status Codes of HyperCat

Code	*Meaning*
200	Success
201	Created
204	No response
204	Successfully Deleted (For DELETE request)
400	Bad request (e.g., malformed input)
401	Unauthorised
404	Not found
409	Conflict (e.g., insert existing href)
501	Not implemented

service uses to receive input, C is the cost of the service, and P is the advertised precision of the service. Each information service type corresponds to an abducible predicate, so we associate services with abducible predicates. Let us note that the advertised precision and cost of the service may be misleading or vary over time.

Since, many services with different precision and cost may provide the same information, it is a challenge to select the best service(s) to contact to retrieve missing information. In the following section, we describe how we effectively select services in order to maximise success rate while minimising cost.

CHAPTER VII

INFORMATION GATHERING

Once we determine the unknown information such as the methane concentration in a specific mine, we may use one or more information services to gather the missing information and resume policy reasoning afterwards.

We focus on efficiently selecting registered external services from our catalogue server. As we described previously, the purpose of the catalogue server is to provide interoperability by using metadata properties. Information services use these service description (SD) properties to register to the catalogue, regardless of their implementation language or their platform. The selection of services determined by the needs (information type), capabilities (input and output methods/formats) of the user platform. If there are several suitable information services that can provide the same necessary information, the best combination of services should be selected among them in order to achieve efficient information gathering system. The *cost* and *precision* values provided by services are helpful (but not necessarily accurate) indicators of effectiveness of services. By using the *cost* and *precision* information, we may effectively select services in order to maximise success rate of the system while minimising the cost.

To achieve our goal, we use Knapsack problem [24] where a Hitch-hiker wants to fill up his knapsack with objects in a way that will give him maximum comfort while not exceeding his knapsack's capacity. Each item has associated size (weight) and value (profit). Sum of the item values that will be inside of the knapsack should be maximised while sum of their weights must be less than or equal to knapsack capacity.

Knapsack problem can be formulated as follows:

$$\begin{aligned} &\text{maximise} \sum_{i=1}^{n} p_i x_i \\ &\text{subject to} \sum_{i=1}^{n} w_i x_i \leq c \end{aligned} \tag{1}$$

$$\tag{2}$$

where

$$\begin{aligned} x_i &\in \{0,1\} \text{ (1 if } i^{\text{th}} \text{ item is selected, 0 otherwise)} \\ p_i &= \textit{profit} \text{ of } i^{\text{th}} \text{ item} \\ w_i &= \textit{weight} \text{ of } i^{\text{th}} \text{ item} \\ c &= \textit{capacity} \text{ of the knapsack} \\ n &= \text{number of items} \end{aligned} \tag{3}$$

The formulated problem is called **0-1 Knapsack Problem** which consists of a positive integer the capacity W and n items. The main characteristic of the problem is that each item is restricted to be selected (x_i) zero or one times. The **Bounded Knapsack Problem** removes the selection restriction but limits the copies of each item (x_i) to a value $c_i : x_i \in \{0, 1, ..., c_i\}$. The **Unbounded Knapsack Problem** removes the limit on the copies of each item but restricts x_i to be a non-negative integer number : $x_i \geq 0$.

There are also many variations of the knapsack problems that occur by changing parameters such as number of items (n), objectives (p and w) and knapsacks (W). Our approach reflects the 0-1 Knapsack problem, we have ***services*** as items, ***cost*** as weight, ***precision*** as profit, ***catalogue*** as item list, a ***cost limit*** as capacity and each item is restricted to be selected zero or one times. Using dynamic-programming approach, knapsack algorithm can be represented as follows;

Algorithm 1 Knapsack Dynamic-Programming Solution

Require: The algorithm takes following inputs; W is the capacity (*cost limit*), n is the number of items (*services*), and the two sequences $p = \{p1,\ p2,\ .\ .\ .\ .\ ,\ p_n\}$ (*precision*) and $w = \{w1,\ w2,\ .\ .\ .\ .\ ,\ w_n\}$ (*cost*).

```
procedure KNAPSACK($p, w, n, W$)
    for $w \leftarrow 0$ to $W$ do
        $c[0, w] = 0$
    end for
    for $i \leftarrow 1$ to $n$ do
        $c[i, 0] = 0$
        for $j \leftarrow 0$ to $W$ do
            if $w[i] \leq j$ then
                $c[i, j] = max(c[i-1, j], c[i-1, j-w[i]] + p[i])$
            else
                $c[i, j] = c[i-1, j]$
            end if
        end for
    end for
end procedure
```

Ensure: At the end of the function, c[n, W] contains the maximum weight that knapsack can hold.

By making use of Knapsack Algorithm 1, we find the list of the most efficient services that would fit in the capacity and use them to retrieve missing information necessary for policy reasoning as we mentioned in previous chapters. After we completed our knowledge base, we can reason whether our policy is violated or not. Next sections will demonstrate case studies regarding service selection and evaluation.

CHAPTER VIII

CASE STUDIES

In this section, we describe our implementation and approach through case studies.

8.1 Privacy Control in Social Networks

Let us assume that the user wants to hide his current location in a social networking environment such as Facebook. Even though social networks provide access settings for hiding location information, if user shares a photo that discloses his/her location, his/her policy may be breached unintentionally. The user's pictures should be examined before shared on-line to see if the user's location can be inferred through these pictures. If a picture reveals the sensitive location information, it may not be shared.

In our first case study, we allow user to select his/her location access policy (Figure 3).

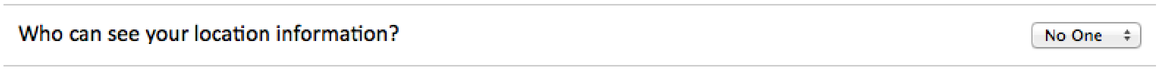

Who can see your location information? No One

Figure 3: Location Access Policy Settings

Based on the settings user selected, we define prolog facts with the information that can be received from social network; in our case Facebook[1]. Facebook provides who liked, commented or tagged-in informations of users' photos. With these information in hand, we have three options for location access policy:

Public: Everybody have access to user's location. Therefore, no need to control accessing location information through policy reasoning.

Friends: Only friends have access to user's location. In this case, we may check what

[1]http://www.facebook.com

are the possible ways for accessing users' location. In Facebook, we need to check whether any of the user's friend interacted (liked, commented or tagged) with user's photo. Since, in Facebook, if anybody interacted with content, their friends also get to see the content. Therefore, if location information can be inferred from photo, third parties could see the location of user. Our knowledge base may encode these rules of the Facebook system as follows:

Facebook domain Rule:

```
canAcess(X,C):-
    hasFriend(X,Y),
    canAccess(Y,C).

canAccess(Y,C):-
    like(Y,C);
    commentTo(Y,C);
    taggedIn(Y,C).
```

User Policy Rule:

```
forbid_nonfriend_accessTo(User, Content):-
    in(User, Content),
    hasLocationInfo(Content),
    nonFriend(User, X),
    canAccess(X, Content).
```

As we mentioned, if user's friend interacted with content and content contains location information, access to the content should be restricted. However, by the nature of social networks, even if there is no interaction, it doesn't mean there will never be. To prevent a violation in the future, the content should be closed to comments, likes, and tagging. Otherwise, as Facebook[2] provides location information of photos, user's location can be accessed by others. On the other hand, even if Facebook doesn't provide the location information, we can not conclude that the content doesn't contain location information; it can be inferred from the content itself, it doesn't need to be represented explicitly.

A user may also want to restrict anybody to see his/her location information. Therefore, we need a policy to forbid access to user's content if content contains location.

```
forbid_any_accessTo(User, Content):-
    in(User, Content),
    hasLocationInfo(Content).
```

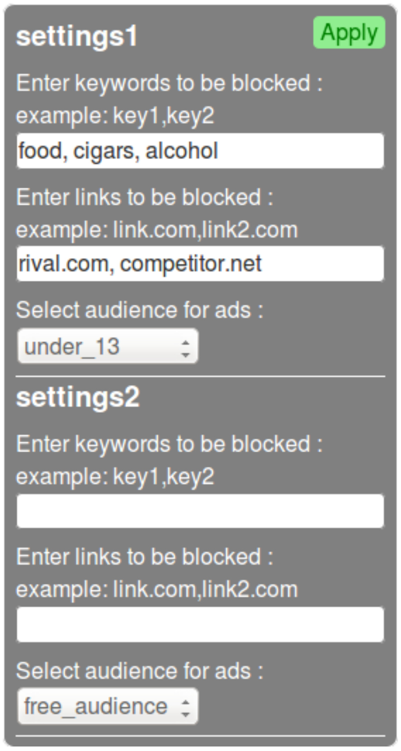
As in the previous option, location information may not be provided by Facebook. By using abductive reasoning and information gathering, we may reason about users' privacy policies event though the knowledge base is incomplete.

8.2 Online Advertisement Control

Let us assume that a website owner has a policy on what kind of advertisements should not be shown on his website, e.g., wants to block his competitors' advertisements. A policy reasoner working on the client side (i.e., web browser) should be used to reason with the owner's policies and block some advertisements. While doing so, the reasoner may use several information services based on the content and the meta information about the advertisements.

In this case study, web sites or content providers, by using our JavaScript library, have options to select keywords, links and targeted audience age (Figure 4) to filter ads displayed on their webpage. On the other hand, if web application provides age of the current web user, we automatically filter the ads based on the age as well.

[2]https://developers.facebook.com/docs/graph-api



Enter keywords to be blocked :
example: key1,key2
food, cigars, alcohol
Enter links to be blocked :
example: link.com,link2.com
rival.com, competitor.net
Select audience for ads :
under_13

Enter keywords to be blocked :
example: key1,key2
Enter links to be blocked :
example: link.com,link2.com
Select audience for ads :
free_audience

Figure 4: Ads Display Policy Settings

Keywords: The user may select keywords to block any advertisement that contains those keywords. For instance, considering the following Prolog rules (*forbid_ads_displayIn*) and facts defined by user's settings (*restrictedIn*), an image containing a smoking person (*relatedTo(content, cigarette)*) violates the first policy rule:

```
% A region on the web site can not contain ads related to
   keywords restricted by user.
 forbid_ads_displayIn(Content, Region):-
     restrictedIn(X, Region),
     relatedTo(Content, X).

% No region on the web site can contain ads related to
   competitor of xyz.com
forbid_ads_displayIn(Content, _):-
```

```
    competitorOf(X,'xyz.com'),
    relatedTo(Content, X).

% region0 on the web site cannot contain ads related to
   food.
restrictedIn(food, region0).
% no region on the web site can contain ads related to
   cigarette.
restrictedIn(cigarette, _).
% no region on the web site can contain ads related to
   alcohol.
restrictedIn(alcohol, _).
```

The difference between what advertisement provider (e.g., Google Adsense [25]) does to filter ads and what we do is that we are not only filtering ads based on the meta-data provided by advertiser, but also the information that can be derived from the content in various ways, e.g., through text or image analysis. We use open-world assumption to avoid relying on the information provided by the advertiser as they may enter incomplete information about ads and this would disrupt the filtering system.

Audience: The user may select the targeted age group to filter ads based on the content. Ads are filtered according to the references in images. When defining age groups and references for each age group, we took “RTUK Smart Signs” [26] as reference. These are representation of the references for age groups :

```
class_references(under_18,Content) :-
    reference(savagery,Content);
    reference(horror,Content);
    reference(violence,Content);
    reference(vandalism,Content);
    reference(sexuality,Content).

class_references(under_13,Content) :-
    class_references(under_18,Content);
    reference(figurative_explanations,Content);
    reference(bad_idols,Content);
    reference(authoritarian_behaviors,Content).

class_references(under_7,Content) :-
    class_references(under_13,Content);
    class_references(under_18,Content);
    reference(fictional_characters,Content);
    reference(character_transformations,Content);
    reference(physical_abuse,Content);
    reference(aggressive_content,Content).
```

For ages above 18, there are no references defined. Thus, no content will be filtered for this age group. The following is the rules for each age group:

```
% Do not display ad if it has specific reference that
% should be avoided for selected age group
%
forbid_ads_displayIn(Content, _) :-
    classification(under_18,Content),
    class_references(under_18,Content).

forbid_ads_displayIn(Content, _) :-
```

```
    classification(under_13,Content),
    class_references(under_13,Content).

forbid_ads_displayIn(Content, _) :-
    classification(under_7,Content),
    class_references(under_7,Content).
```

As mentioned above, we also provide automatic filtering based on user's age. If web site provides user's age, we find the user's age group and apply filtering based on his age group:

```
%
% Automatic age classification based on user's age
%
forbid_ads_displayIn(Content, _) :-
    user_age(Age),
    Age<7,
    class_references(under_7,Content).

forbid_ads_displayIn(Content, _) :-
    user_age(Age),
    Age<13,
    class_references(under_13,Content).

forbid_ads_displayIn(Content, _) :-
    user_age(Age),
    Age<18,
    class_references(under_18,Content).
```

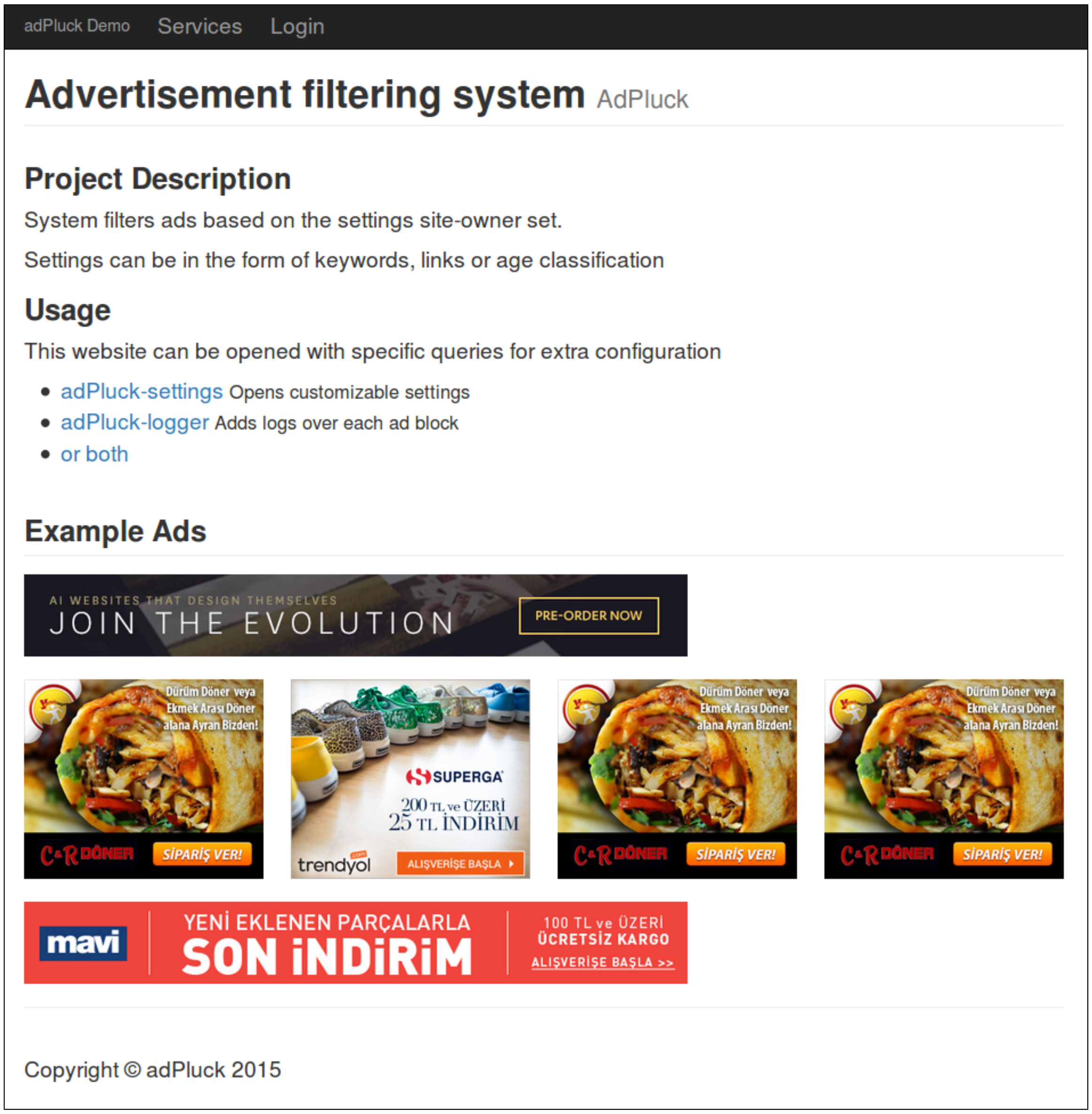
The references mentioned in rules don't exist in our knowledge base as they are not provided by advertisement providers. Therefore, we are not be able to run a query and conclude a decision by default. To draw accurate conclusions for our policy rules, we use various services and find unknown information. We complete our knowledge base in order to make best decision.

On the other hand, the advertiser may provide some meta information about the advertisements, i.e., the content may contain keywords or category information. To reason with policy, the meta information and the multimedia (e.g., pictures and videos) should be extracted from content, e.g., if there exists adult material on the images on the advertisement.

Several simple information services are implemented to process multimedia represented in the content. For instance, an information service may check if an image contains a cigarette brand. This service receives image url as input and finds all the texts in image using optical character recognition (OCR). It contains a list of all cigarette brands and check if any of these brands appear in the extracted text. Hence, this simple service can be used to check if an advertisement contains a cigarette brand.

Meta information for this service contains its cost and precision in HyperCat properties. Precision value is the indication of service's performance in returning accurate results where 10 being the most successful. On the other hand, cost consists of collective efforts for using the service. These efforts may include run time, response time, required resources, and so on.

To exhibit this case study, we implement a system that filters ads based on the policies of the hosting web site. We created a sample web site that hosts sample advertisement images. Figure 5 demonstrates the layout of the web page. We also created a JavaScript library, namely adPluck, that handles advertisement filtering functionalities. We mentioned the display settings (in Figure 4) and how we use the selected options to create our Prolog rules and policies.



adPluck Demo Services Login

Advertisement filtering system AdPluck

Project Description

System filters ads based on the settings site-owner set.

Settings can be in the form of keywords, links or age classification

Usage

This website can be opened with specific queries for extra configuration

- adPluck-settings Opens customizable settings
- adPluck-logger Adds logs over each ad block
- or both

Example Ads

AI WEBSITES THAT DESIGN THEMSELVES
JOIN THE EVOLUTION
PRE-ORDER NOW
Dürüm Döner veya Ekmek Arası Döner alana Ayran Bizden!
C&R DÖNER
SİPARİŞ VER!
SUPERGA
200 TL ve ÜZERİ 25 TL İNDİRİM
trendyol
ALIŞVERİŞE BAŞLA
mavi
YENİ EKLENEN PARÇALARLA
SON İNDİRİM
100 TL ve ÜZERİ ÜCRETSİZ KARGO
ALIŞVERİŞE BAŞLA >>

Copyright © adPluck 2015

Figure 5: AdPluck Example Layout

We let web site administrator to define multiple settings for different advertisement blocks, which are implemented as DIV tags. As seen in Figure 4, there are two different settings defined. The administrator may apply these settings to advertisement blocks by assigning “data-adpluck” custom data attribute as in the following lines;

```html
<div id="ads1" class="col-lg-12" data-adpluck="settings1">
    <a href="http://bit.ly/1RmnUT" data-metadata="Website design">
        <img class="img-responsive portfolio-item" src="ads/639x78/14756452170571942819.png" alt="">
    </a>
</div>

<div id="ads2" class="col-sm-3 col-xs-6" data-adpluck="settings2">
    <a href="http://tr.alpha-wars.com/?r=gogaw1trb" data-metadata="Yemeksepeti durum doner">
        <img class="img-responsive portfolio-item" src="ads/300x250/4053084747909437676.jpg" alt="">
    </a>
</div>
```

The “data-adpluck” custom attribute can be assigned to any advertisement block. E.g., Google Adsense uses a JavaScript code to fetch advertisement content and displays contents in iframes, “data-adpluck” attribute can be assigned to those iframes or a wrapper element. Our adPluck library parses webpage for this specific attribute and apply filters based on attribute value (i.e., settings name) and selected options.

Our library starts parsing the webpage after a few seconds as some advertisement providers (e.g., Google Adsense) loads their content dynamically. In order not to miss any content and detect content change, we set a timer to control content of the advertisement wrapper. If content has changed since last check, we start over the filtering process.

Our process starts with information gathering from webpage. Links, images and keywords are parsed from blocks to create our knowledge base. Keywords may be represented in content in different forms such as descriptive texts or attributes (e.g., ids, classes, names) that defines advertisement block. Different keyword parsing techniques should be implemented for different content providers. As proof-of-concept (POC), we defined our own "data-metadata" custom attribute to represent keywords related to the advertisement. As seen in above code block, different descriptive keywords are defined for each advertisement block.

The gathered information is used as facts, which are asserted to our Prolog knowledge base. As mentioned previously, by using tuProlog's multi-programming approach, we implemented a Java class to run Prolog queries. Our Java implementation will handle assertion of the facts (gathered information), rules and policies. It also uses abductive reasoning to figure out which information is missing in the knowledge base to reason with the policies.

The facts are asserted to the knowledge base using *asserta* built-in predicate [27] of tuProlog. The '*asserta(Clause)*' predicate adds *Clause* to the beginning of the database. Based on settings displayed in Figure 4 and the advertisement blocks represented above, created assertion commands are as follows:

```
{
    "assertions":[
        "asserta(restrictedIn(food,block0))",
        "asserta(restrictedIn(cigars,block0))",
        "asserta(restrictedIn(alcohol,block0))",
        "asserta(relatedTo(content0,website))",
        "asserta(relatedTo(content0,design))",
        "asserta(restrictedIn('rival.com',block0))",
        "asserta(restrictedIn('competitor.net',block0))",
        "asserta(relatedTo(content0,'http://bit.ly/1RmnUT
           '))",
        "asserta(classification(under_13,content0))",
        "asserta(user_age(17))",
        "asserta(classification(free_audience,content1))",
    ]
}
```

The *block0* and *block1* are references to the advertisement regions. The keywords '*website*' and '*design*', parsed from *block0*'s 'data-metadata' attribute, are asserted to the knowledge base as well as the links that *block0* contains. Even if *block1* has also link and keywords defined in its 'data-metadata' attribute, we don't assert them as user did not define keyword and link filtering for 'settings2'.

As represented above, we create 'forbid_ads_displayIn' rules to filter keywords and links based on the settings user selected in his policy settings (Figure 4). The *restrictedIn* predicate (created based user's filtering settings) will be matched with our knowledge base (represented by *relatedTo*) and if any of our rules hold, this would mean that we need to filter respective advertisement block.

On the other hand, as advertisement providers may not provide sufficient information (e.g., references in images), we need to complete our knowledge base in order to filter advertisements successfully. To inform our application that some information needs to be known in order to make accurate decisions, we define abducible predicates;

```
abducible(reference(X,Y)).
abducible(relatedTo(X,Y)).
```

In our sample application, the predicates *reference*, *relatedTo* have been defined as abducibles. Any lack of information in these types will be considered as unknown instead of false. We will then use external information services to gather information about these unknown information.

The execution of policies is also handled by making use of tuProlog's Java integration. We call the Java class implementation with the query, it then uses tuProlog to run Prolog and execute our policy. The policy execution query for content (*content0*) in region (*block0*) is;

```
abduce(forbid_ads_displayIn(content0, block0),S).
```

As we mentioned in Chapter 5, *abduce* predicate represents our abduction algorithm and *forbid_ads_displayIn* represents our policy rules as represented above. After the execution of abduction algorithm, it returns the list of solutions necessary to reason about policies. If returned list is empty, this means that abduction algorithm is already able to reason with policies with the current information in knowledge base. If solution list is not empty, first we convert the solution list to JSON format in our Java class to easily interpret solutions in our JavaScript library. The result of the abduction for our query is as follows:

```
{
    "solutions":{
        "0":["relatedTo(content0,'competitor.net')"],
        "1":["relatedTo(content0,'rival.com')"],
        "2":["relatedTo(content0,alcohol)"],
        "3":["relatedTo(content0,cigars)"],
        "4":["relatedTo(content0,food)"],
        "5":[
            "reference(authoritarian_behaviors,content0)",
            "reference(bad_idols,content0),reference(
               figurative_explanations,content0)",
            "reference(sexuality,content0),reference(
               vandalism,content0),reference(violence,
               content0)",
            "reference(horror,content0),reference(savagery
               ,content0)"
        ],
        "6":[
            "reference(sexuality,content0)",
            "reference(vandalism,content0)",
            "reference(violence,content0),reference(horror
               ,content0)",
            "reference(savagery,content0)"
        ]
    }
}
```

After the creation of the possible policy violation explanations, information gathering system takes the solutions as input and tries to prove them by gathering missing information. We have variety of services that provide missing information required to reason with policies. Each service is registered to the catalogue server with information such as location, information type, input type and input variable information, which are used while utilizing the service. Also, precision and cost information is used while selecting services, content type information is used while processing data provided by service(See Tables 1 and 2).

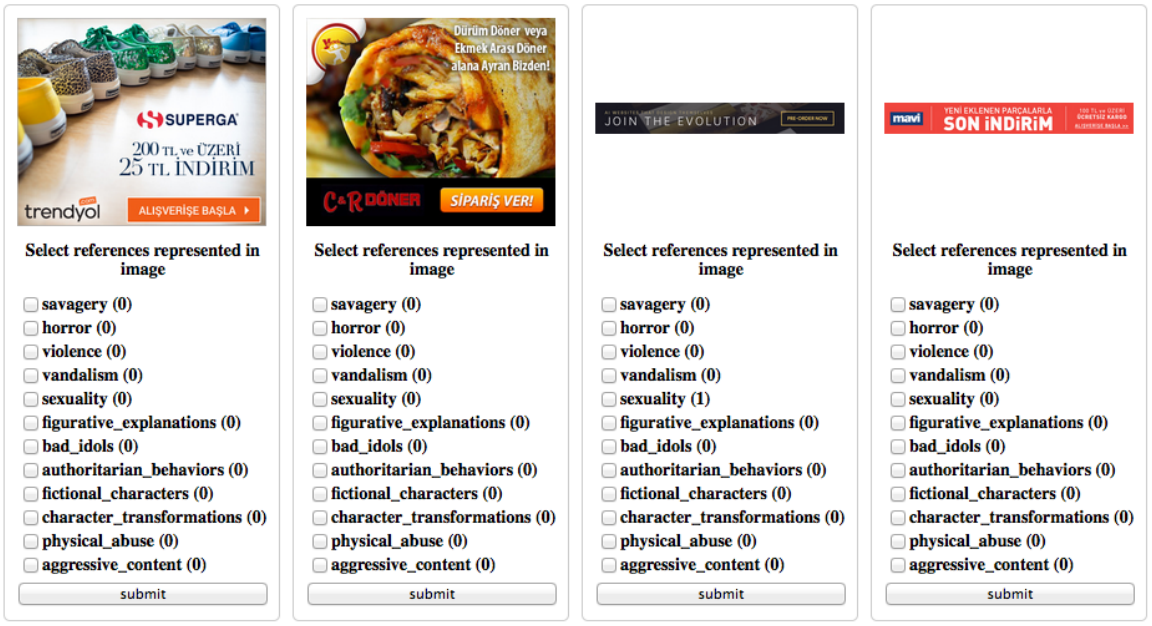
Our library parses solution list and finds missing information that will be requested from services. Considering our solution list, we need services that provides information about *link*, *keyword* and *reference*. We use HyperCat Pathfinder to store links and informations of services as described before. Our application selects best services from the catalogue in the means of precision and cost using the knapsack algorithm. Normally, any developer can implement their own services and register to our catalogue server. In this study, we created sample services for each information type:

```
[
    {
        "href": "http://54.187.189.6/services/url_redirect_finder.php",
        "i-object-metadata": [
            {
                "rel": "urn:X-tsbiot:rels:hasDescription:en",
                "val": "Finds redirections in URL"
            },
            {
                "rel": "urn:X-tsbiot:rels:isContentType",
                "val": "text/plane"
            },
            {
                    "rel": "urn:X-tsbiot:rels:informationType"
                       ,
                    "val": "link"
               },
               {
                    "rel": "urn:X-tsbiot:rels:inputType",
                    "val": "GET"
               },
               {
                    "rel": "urn:X-tsbiot:rels:inputVariable",
                    "val": "URL"
               },
               {
                    "rel": "urn:X-tsbiot:rels:precision",
                    "val": "8"
               },
               {
                    "rel": "urn:X-tsbiot:rels:cost",
                    "val": "6"
               }
          ]
     },
     {
          "href": "http://54.187.189.6/services/tesseractOCR
             /keyword_finder.php",
          "i-object-metadata": [
               {
                "rel": "urn:X-tsbiot:rels:hasDescription:en",
                "val": "Finds keywords in image"
            },
            {
                "rel": "urn:X-tsbiot:rels:isContentType",
                "val": "text/csv"
            },
            {
                "rel": "urn:X-tsbiot:rels:informationType",
                "val": "keyword"
            },
            {
                "rel": "urn:X-tsbiot:rels:inputType",
                "val": "GET"
            },
            {
                "rel": "urn:X-tsbiot:rels:inputVariable",
                "val": "URL"
            },
            {
                "rel": "urn:X-tsbiot:rels:precision",
                "val": "8"
            },
            {
                "rel": "urn:X-tsbiot:rels:cost",
                "val": "3"
            }
        ]
    },
    {
        "href": "http://54.187.189.6/services/userReference/reference_finder.php",
        "i-object-metadata": [
            {
                "rel": "urn:X-tsbiot:rels:hasDescription:en",
                "val": "Asks people for references in images"
            },
            {
               "rel": "urn:X-tsbiot:rels:isContentType",
               "val": "text/csv"
            },
            {
               "rel": "urn:X-tsbiot:rels:informationType",
               "val": "reference"
            },
            {
                "rel": "urn:X-tsbiot:rels:inputType",
                "val": "GET"
            },
            {
                    "rel": "urn:X-tsbiot:rels:inputVariable",
                    "val": "URL"
                },
                {
                    "rel": "urn:X-tsbiot:rels:precision",
                    "val": "8"
                },
                {
                    "rel": "urn:X-tsbiot:rels:cost",
                    "val": "10"
                }
            ]
    }
]
```

url_redirect_finder: Links used in advertisements are usually redirected because of tracking purposes. This complicates the detection and filtering process for urls. We created this service to find actual url in case of redirection. The url_redirect_finder service gets url and uses PHP[3] to fetch headers of HTTP request. It looks for any redirected location and returns the actual url in "text/plane" format.

keyword_finder: Not all keywords related to the advertisement are represented in the text format in advertisement blocks. As we mentioned above, we parse 'data-metadata' attribute to create our knowledge base. However, more information can be extracted from images. This service gets image url and uses 'tesseract-ocr' to parse keywords in images. Tesseract[4] is considered to be one of the most accurate multi-platform open source optical character recognition (OCR) engine. After downloading

[3]https://php.net/manual/kr/function.get-headers.php
[4]https://code.google.com/p/tesseract-ocr/


SUPERGA
200 TL ve ÜZERİ 25 TL İNDİRİM
trendyol
ALIŞVERİŞE BAŞLA
Dürüm Döner veya Ekmek Arası Döner alana Ayran Bizden!
C&R DÖNER
SİPARİŞ VER!
JOIN THE EVOLUTION
mavi
YENİ EKLENEN PARÇALARLA SON İNDİRİM
submit

Select references represented in image	Select references represented in image	Select references represented in image	Select references represented in image
savagery (0)	savagery (0)	savagery (0)	savagery (0)
horror (0)	horror (0)	horror (0)	horror (0)
violence (0)	violence (0)	violence (0)	violence (0)
vandalism (0)	vandalism (0)	vandalism (0)	vandalism (0)
sexuality (0)	sexuality (0)	sexuality (1)	sexuality (0)
figurative_explanations (0)	figurative_explanations (0)	figurative_explanations (0)	figurative_explanations (0)
bad_idols (0)	bad_idols (0)	bad_idols (0)	bad_idols (0)
authoritarian_behaviors (0)	authoritarian_behaviors (0)	authoritarian_behaviors (0)	authoritarian_behaviors (0)
fictional_characters (0)	fictional_characters (0)	fictional_characters (0)	fictional_characters (0)
character_transformations (0)	character_transformations (0)	character_transformations (0)	character_transformations (0)
physical_abuse (0)	physical_abuse (0)	physical_abuse (0)	physical_abuse (0)
aggressive_content (0)	aggressive_content (0)	aggressive_content (0)	aggressive_content (0)

Figure 6: AdPluck 'reference_finder' Service

image and using tesseract-ocr, recognized keywords are returned in the "text/csv" format.

reference_finder: This service gets url of the image, saves image url to its database and then uses crowdsourcing to ask people to categorise images based on RTUK references. We implemented a basic webpage to show images and provide reference selection form for users to make choices (Figure 6). This service is considered to be a very costly process as it needs to wait for user input before responding to the application and it relies on a manual process instead of an automatic reference detection. The service waits for few seconds for user input and then returns references, which users selected, to application in "text/csv" format.

As we have only one service for each information type in our hypercat server, our knapsack algorithm always selects represented services for our information gathering system when necessary. In our case, we use all three services as we need to get information about *link*, *keyword* and *reference*. All services get urls as input. Therefore, for each advertisement block, we send respective advertisement urls to services. We

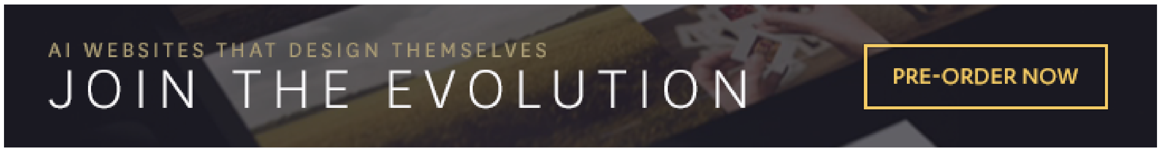


Figure 7: Adpluck Advertisement Sample

will now show results of services for advertisement image in Figure 7, whose wrapper id is "ads1" in HTML block. This image is actually used as advertisement by Google Adsense.

Firstly, we use *url_redirect_finder* service to find whether advertisement link redirects to 'competitor.net' or 'rival.com'. The advertisement url is already parsed and asserted to the knowledge base. However, it does not directly match to the links specified by user. We need to find if "http://bit.ly/1RmnUT" redirects. We send a GET request to the service with the url. Service finds actual url and sends the information to the application. Table 5 shows service's responses to few sample urls.

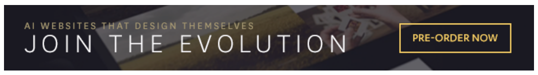
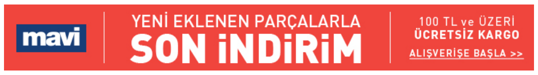
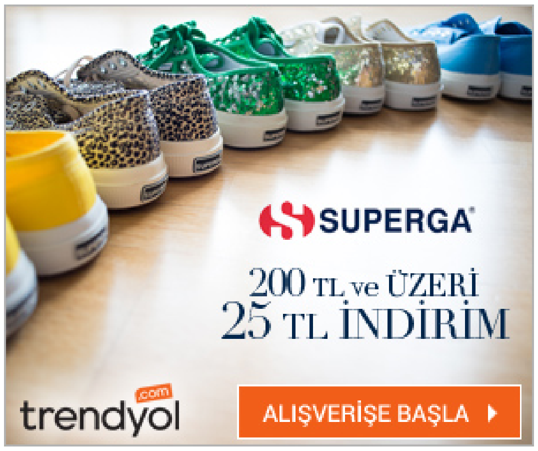
Table 5: *url_redirect_finder* Sample Results

Input	*Output*
http://bit.ly/1RmnUT	http://google.com
http://facebook.com	http://facebook.com
http://goo.gl/Dgr9aX	http://www.google.com/

As seen in Table 5, the redirected url for 'http://bit.ly/1RmnUT' is 'http://google.com' and it still does not match the links user selected to filter. Therefore, no precaution has to be taken for this particular case. If it would match either 'competitor.net' or 'rival.com', we had to hide/block advertisement to fulfil user policy.

Secondly, we use *keyword_finder* to find keywords that are not explicitly represented (i.e., in text format) in the content, but may be extracted from image. We send the url of the image to the service and service extracts the keywords from image using OCR. Table 6 shows service's keyword extractions from few sample images.

Table 6: *keyword_finder* Sample Results

Input	*Output*
 AI WEBSITES THAT DESIGN THEMSELVES JOIN THE EVOLUTION PRE-ORDER NOW	m, webshes, mu, uesmn, ihemselves, join, the, evolution, pre-order, now
 mavi YENİ EKLENEN PARÇALARLA SON iNDiRiM 100 TL ve ÜZERİ ÜCRETSİZ KARGO ALIŞVERİŞE BAŞLA >>	yeni, eklenen, parcalarla, -, sou, indirim, ”um, tl, {e, uzeri, ucretsiz, kargo, alisverise, basla
 SUPERGA 200 TL ve ÜZERİ 25 TL İNDİRİM trendyol .com ALIŞVERİŞE BAŞLA	supehga', gm, tlye, uzer'i, 2;), ti; indirim, r, trendyol

Our service is able to extract texts *'m, webshes, mu, uesmn, ihemselves, join, the, evolution, pre-order, now'* from advertisement image and none of them matches the keywords user selected to filter (alcohol, cigars, food). Therefore, advertisement doesn't need to be blocked for keyword filtering.

Thirdly, we use *reference_finder* service to get references in advertisement and filter them if image contains "authoritarian_behaviors, bad_idols, figurative_explanations, sexuality, vandalism, violence, horror, savagery" references. We send the image url as input and service searches its database for references of stated image. If any reference found for image, service responds with the reference list. Otherwise, service saves image to its database and waits 10 seconds for human evaluation before responding. This is the sample response returned from *reference_finder* service;

```
{
        "savagery":false,
        "horror":false,
        "violence":false,
        "vandalism":false,
        "sexuality":true,
        "figurative_explanations":false,
        "bad_idols":false,
        "authoritarian_behaviors":false,
        "fictional_characters":false,
        "character_transformations":false,
        "physical_abuse":false,
        "aggressive_content":false
}
```

According to the references returned from service, advertisement image contains sexuality references. Somebody informed service's system that the image contains sexuality references. Therefore, based on the site owner's audience selection in policy settings (Figure 4) and the age of the user (17), we need to hide advertisement blocks that contains sexuality reference. Hiding advertisement can be established in a few ways such as removing wrapper div, setting visibility of ads block to hidden, or adding an overlapping image stating that the content is blocked.

The services we talked about are implemented as POC, they may not be accurate. As we mentioned earlier, our purpose is to create a framework that mainly focuses on open-world reasoning and gathering information using external services to complete the knowledge base for best decision making. In this example, we created our initial knowledge base, defined our policies, run policies and found missing information using abductive reasoning, used external services to complete missing information and finally filtered advertisement blocks which contradicts policies site-owner set.

CHAPTER IX

EVALUATIONS

As we mentioned in previous chapters, the selection of information services plays a vital role in the performance of the system. If the selection of services is executed randomly, the high-cost, low-precision information services may be selected frequently and this would lead to wrong results at high cost. We run simulations to indicate the draw backs of random service selection. The simulation consists of 100 steps, in each step a service is selected randomly; independent from it's properties. Table 7 shows select counts of services during test. Service names represents their cost and precision values, e.g., service90_1 indicates 90% success rate with cost 1. As we can see from Table 7, there isn't big difference between selection counts of services, we equally selected high-cost, low-precision services as well as low-cost, high-precision services.

Table 7: Random Selection - Service Select Count

Service	*Count*	*Success*	*Fail*	*Cost*
service90_1	10	10	0	10
service90_3	14	12	2	42
service80_2	10	8	2	20
service80_1	12	9	3	12
service90_10	12	7	5	120
service70_3	10	8	2	30
service40_1	7	4	3	7
service50_4	3	1	2	12
service30_2	11	4	7	22
service20_3	11	2	9	33
Total	100	65	35	308

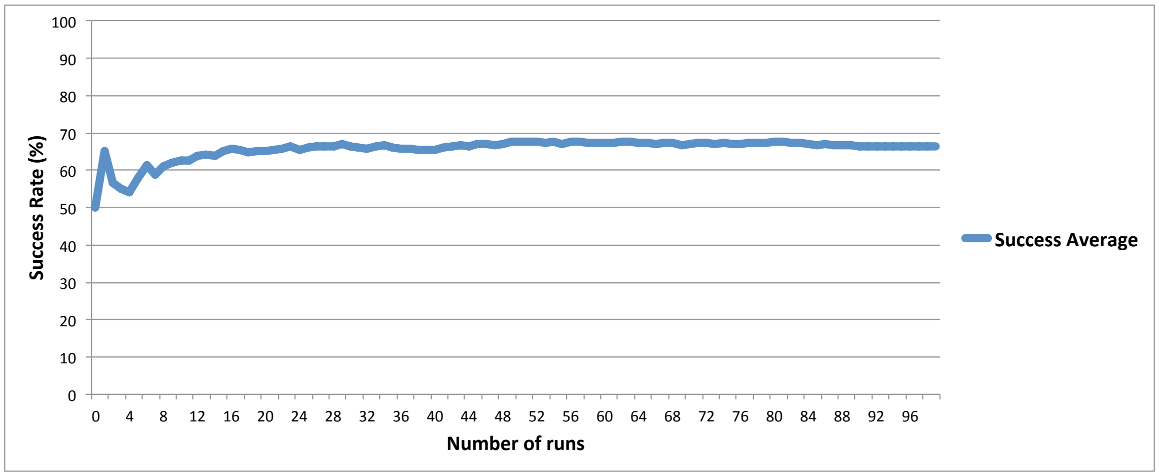


Figure 8: Random Service Selection Success Average

Figure 8 shows success rate of random selection. The uncontrolled selection of services concluded on success rate close to the average success rate (64%) of the services. As presented in Table 7, 3,08 average cost is also similar to the average cost value (3) of services.

9.1 Service Selection

To increase the success rate of the policy reasoning by efficiently selecting services, we use knapsack as we mentioned in Section 2.3. We now show the evaluation of the service selection of the following policy:

```
A, B, C -> D
R -> C
H -> C
```

Where all predicates are defined as abducibles. Using abduction algorithm, the result for the query **abduce(D,[])** is

```
{A,B,R}
{A,B,H}
```

Table 8: Service Selection - Service Properties

Service	*Precision*(%)	*Cost*
service99_1	99	1
service90_3	90	3
service90_10	90	10
service80_4	80	4
service70_4	70	4
service60_2	60	2
service50_4	50	4
service40_1	40	1
service30_2	30	2
service20_3	20	3
service1_1	1	1

For each predicate (A,B,R and H) we created services with identical properties as in Table 8. Therefore, for each abducible predicate, we select external services to gather information about them. Using knapsack formulation represented in Section 2.3;

$$\begin{aligned} n &= 11 \\ (p_i) &= (99, 90, 90, 80, 70, 60, 50, 40, 30, 20, 1) \\ (w_i) &= (1, 3, 10, 4, 4, 2, 4, 1, 2, 3, 1) \\ c &= 7 \end{aligned} \tag{4}$$

The optimal solution is

$$\begin{aligned} x &= (1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1) \\ z &= 290 \\ s &= 5 \end{aligned} \tag{5}$$

As seen above, for each predicate, 5 services are selected for information gathering; *service99_1, service90_3, service60_2, service40_1, service1_1.* Now, we represent how we combine results of the services in order to achieve most accurate results.

9.2 Result Assessment

As we may select multiple information services for decision making process, we need to accurately combine information from each selected service. In our setting, information provided by services are binary truth values (either true or false) for grounded predicates in policies. The grounded predicates are actually binary propositions such as *friend(john, jane)*. For a specific proposition, we may query more than one service; each reports its estimate for the truth value of the proposition. There are few approaches to combine result of the services;

Optimstic : Having one of the services return true is enough to conclude that the proposition is true.

Pessimistic : Having one of the services return false is enough to conclude that the proposition is false.

Both Optimistic and Pessimistic approaches consider neither the precision nor the cost of the services. Therefore, they are not good for finding an optimum solution for service selection. For example, consider we are using optimistic approach, if a low precision service returns true, we assume the proposition is true even if all other, possible higher precision, services return false.

Simple Voting : Each selected service has one vote and result is determined with consensus. This approach is more reliable than previous approach as it considers reports from all services during fusion. On the other hand, simple voting does not take precision of services into account.

Weighted Voting : In order to consider precision, we propose weighted voting where votes of services are weighted based on their precisions. Higher precision services will have more affects in result.

To test the accuracy of these approaches, we created a test on policy represented above. We assigned probability values to policy results such as; 0.3, 0.5 and 0.7. These values indicate the probability of policy to hold. If policy D holds, either $\{A,$

Table 9: Policy Result Assessment - Predicate Assignments

	D	A	B	R	H
P_1	T	T	T	T	F
P_2	T	T	T	F	T
P_3	T	T	T	T	T
P_4	F	F	T	T	T
P_5	F	T	F	T	T
P_6	F	F	F	T	T
P_7	F	F	F	F	F

B, R } or {*A, B, H* } should hold. Otherwise, both {*A, B, R* } and {*A, B, H* } should fail. To test external services in predicate level, we created predicate lists (represented in Table 9) and assigned values to predicates for both policy's hold and fail cases. Each predicate list has probability to be selected as formulated below.

$$\begin{aligned}
P_t &= \textit{probabilty} \text{ of predicate to be selected when Policy holds}\\
P_f &= \textit{probabilty} \text{ of predicate to be selected when Policy fails}\\
P_p &= \textit{probabilty} \text{ of Policy to hold}\\
i &= (1,2,3,4,5,6,7) \text{ number of predicate list}\\
&\text{If Policy holds}\\
P_i &= P_p * P_t\\
&\text{else}\\
P_i &= (1 - P_p) * P_f
\end{aligned} \tag{6}$$

If probability of policy to hold is *0.3*;

$$\begin{aligned}
P_t &= 0.33\\
P_f &= 0.25\\
P_p &= 0.3
\end{aligned} \tag{7}$$

The probability of P_1, P_2 or P_3 to be selected is *0.099* and the probability of P_4, P_5, P_6 or P_7 is *0.175*. Selected external services and the approach used to combine

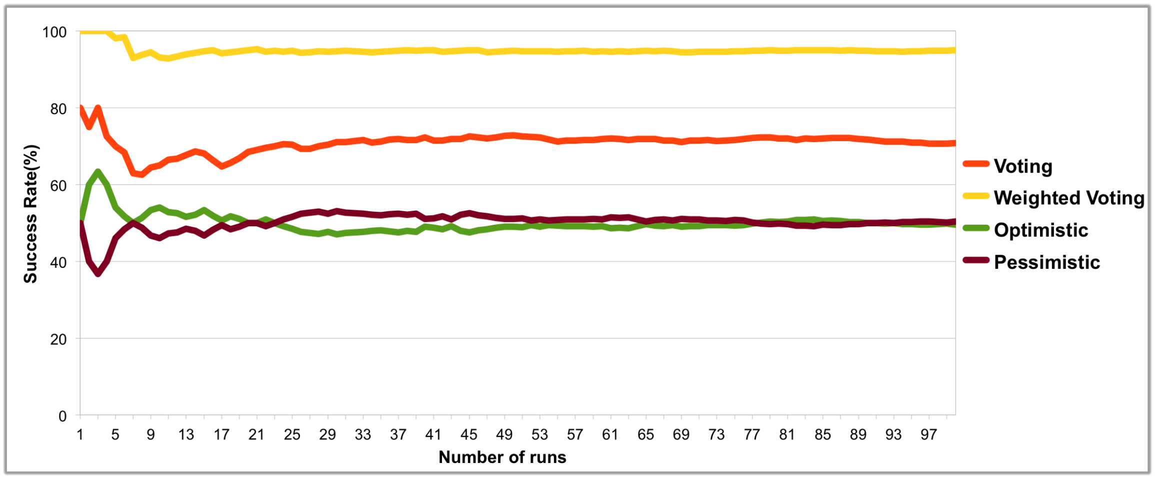


Figure 9: Average Success Rate For Combination Approaches

results should provide these expected results. In predicate level, the evaluation of approaches is represented in Figure 9. Based on the selected predicate list P_i (Table 9) and using the selected services, we calculated success rates of approaches in matching the expected predicate values over 100 runs.

In policy level, Figure 10 shows the average success rates of the approaches for different policy probabilities after 100 runs. On each run, we check whether or not policy (holds or fails depending on the probability) matches the evaluation of the $\{A, B, R\}$ and $\{A, B, H\}$.

As seen in Figure 10, the *voting* approach's success rate decreases as policy hold rate increases. The main reason of this is the difference between maximum negligible wrong predicate amount when policy fails or holds. When policy holds, $\{A, B, R\}$ or $\{A, B, H\}$ should hold. Therefore, maximum 1 predicate can be found wrong; R or H, otherwise policy can not hold. On the other hand, when policy fails, both $\{A, B, R\}$ and $\{A, B, H\}$ must fail together. Therefore, the maximum number of wrong predicates is 2. If A, B, R or A, B, H found wrong (*true*) together, policy holds. There is more room to failure for the tests where policy fails. Therefore, *voting* is more successful when policy expected to fail.

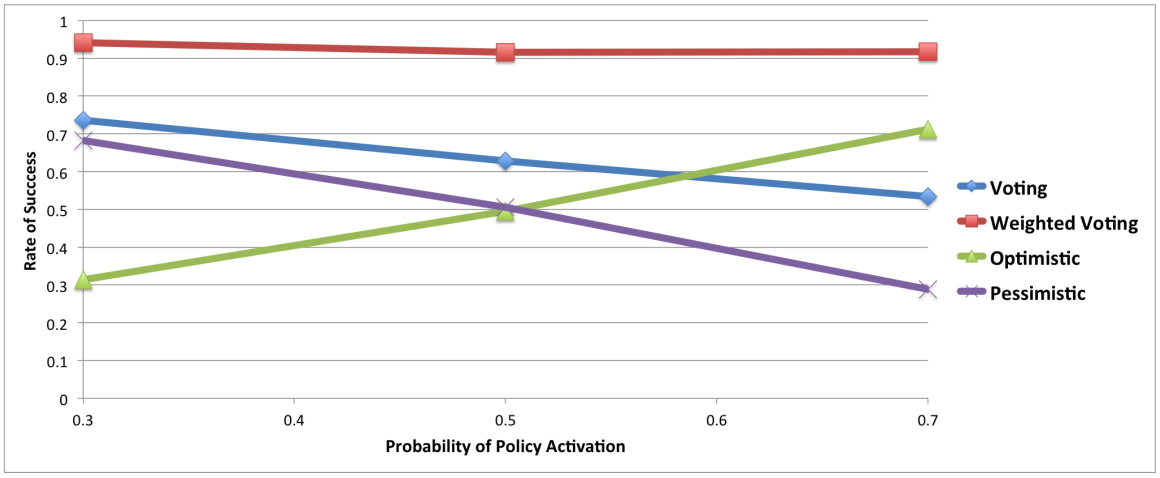


Figure 10: Policy Result Assessment

The *optimistic* and *pessimistic* approaches decreases and increases oppositely. When policy hold rate increases, optimistic approach increases as only one *true* information from services is enough to conclude predicate as *true*. On the contrary, pessimistic approach decreases as only one *false* information is enough to conclude predicate as *false*. Therefore, they are not reliable approaches as hold rate of the policies are variable.

On the other hand, the weighted voting approach achieved to provide most accurate combination for external information services independent from expected results of policy. With combination of knapsack algorithm and weighted voting approach, system achieved more than 90% success rate with 8 cost while the average values of selected services are 55,45% precision and 15,9 cost.

This test highly depends on the precision and cost values provided when services registered to database. These values may be inaccurate or may change over time. Therefore, the cost and precision values should be evaluated as we select and use services.

9.3 Cost and Precision Modelling

Earlier we mentioned the technique we use to select services. However, this method strongly relies on the cost and precision provided by service administrators to efficiently select services. We can not completely rely on the data provided by services. We need to figure out a way to validate both cost and precision values. We also need to consider fall-backs this validation process may lead to.

Services' cost or precision may change overtime (i.e., regression or improvement of service). We may miss efficient services because of the static values, or we may continue to select inefficient services just because they were efficient when they first registered. There are few fall-backs that may occur during validation process. One possibility is that, we may select too many services with low precision values and this could lead to false interpretations in our decision making. At the same time, we may select high cost services and this would decrease the performance of our system.

This is a dilemma called exploration-exploitation trade-off [28]. ***Exploration*** includes things decided by searching, risk taking and experimenting. ***Exploitation*** includes things selected by efficiency, experience and productivity. Exploration is risky while making decisions but may provide benefits in long-term. On the other hand, Exploitation provides benefits in short-term based on the experiences. As in our case, the balance between exploration and exploitation should be adjusted carefully. Using too much exploration may decrease the performance of the system while using only exploitation may prevent system enhancement in long-term.

One possible approach is to run services for a certain number of times with known data in order to calculate their average cost and precision. Figure 11 shows the precision value average for services over 100 runs. Each line represents precision values of services as indicated in legend. These services are sample services that returns binary (*true* or *false*) information based on their precision values as indicated in their name. For example, service90_10 indicates 90% success rate with cost 10.

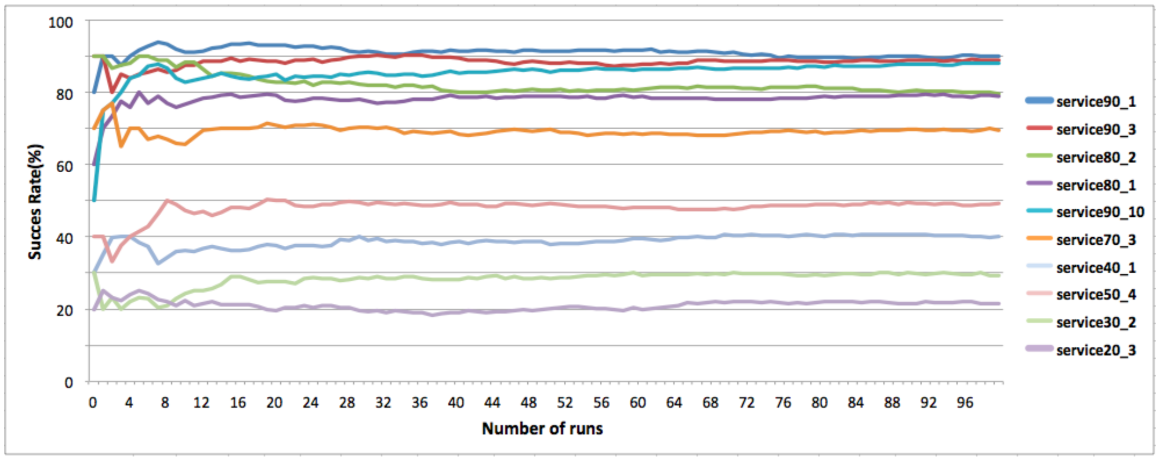


Figure 11: Iterative Service Precision Validation

Even though this is a trivial solution, it is costly. On the other hand, we need to run this iterative validation process periodically to be able to detect value changes. This is an overwhelming process that took so much computation time and it has no direct connection with our decision making process.

We have to consider an approach that is not overwhelming and at the same time useful. In this setting, we calculate average precision and cost of services as we use them. In previous setting, services that are registered with high costs and low precisions are never selected. To prevent that, we decrease the known cost value (in a constant rate) of services that are not selected on each run. This way, costly services get the chance to be selected and if their cost or precision values are changed, we use these values to calculate their average cost and precision. As we discussed earlier, frequently selecting costly services may decrease performance of the system. Exploration factor should be adjusted carefully to balance exploration and exploitation. In addition to Knapsack algorithm, we decrease cost of the services after each run;

$$r = \textit{number of runs}$$

$$e = \textit{exploration factor}$$

$$s_n = \textit{success count of service}$$

$$f_n = \textit{fail count of service}$$

$$c_r = \textit{cost of service on each run}$$

After each run: (8)

If service succeeds : $s_n = s_n + 1$

If service fails : $f_n = f_n + 1$

$$p_i = s_n/(s_n + f_n)$$

$$w_i = (\sum_{i=1}^{r} c_r)/r$$

If $x_i = 0$ *(item is not selected)*

Then $w_i = w_i - e$

Using the same sample services in previous setting and Knapsack formulation;

$$\begin{aligned} n &= 11 \\ (p_i) &= (99, 90, 90, 80, 70, 60, 50, 40, 30, 20, 1) \\ (w_i) &= (1, 3, 10, 4, 4, 2, 4, 1, 2, 3, 1) \\ c &= 7 \end{aligned} \tag{9}$$

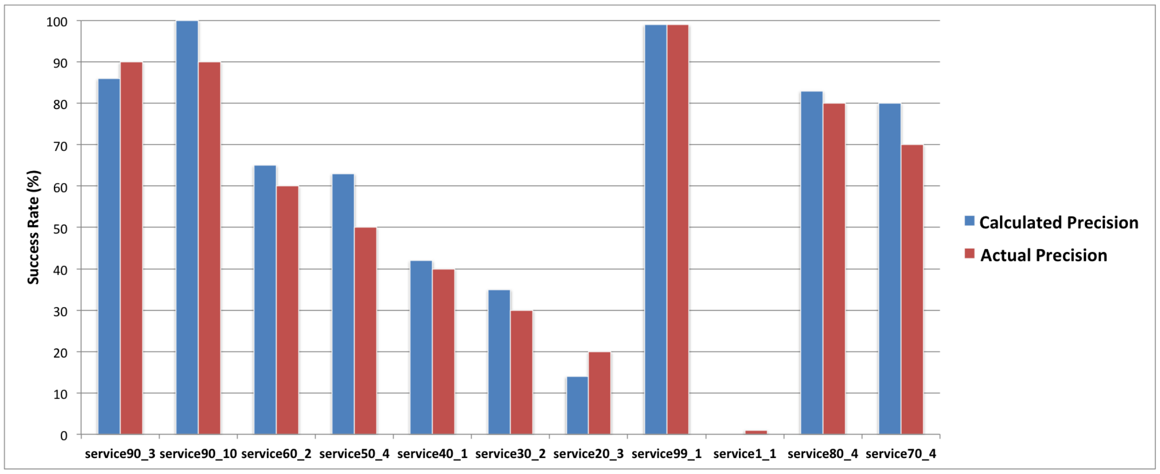


Figure 12: Actual and Calculated Service Precisions with Knapsack Selection

The optimal solution after first run;

$$x = (1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1)$$
$$z = 290$$
$$s = 5 \tag{10}$$

Calculated costs after service selection

$$w_i = (1, 3, 9.9, 3.9, 3.9, 2, 3.9, 1, 1.9, 2.9, 1)$$

Eventually, as cost values of unselected services decreases, all services will be selected by knapsack algorithm. If any service's cost or precision changes over time, we will be able to calculate their properties successfully by matching the selected services' results to Table 9 and this will improve our service selection accuracy in the means of cost and precision. To test success rate of the system, we initially set cost and precision values of each service to 0 and 10, respectively. Figure 13 represents the precision values for p_i over each run and Figure 12 represents the comparison between actual and calculated precision values for each service.

The horizontal line parts in Figure 13 represents the runs in which relative service is not selected because of its high cost value. Earlier we mentioned that the cost may be a collection of fallbacks. As POC, we only used service run time as cost in

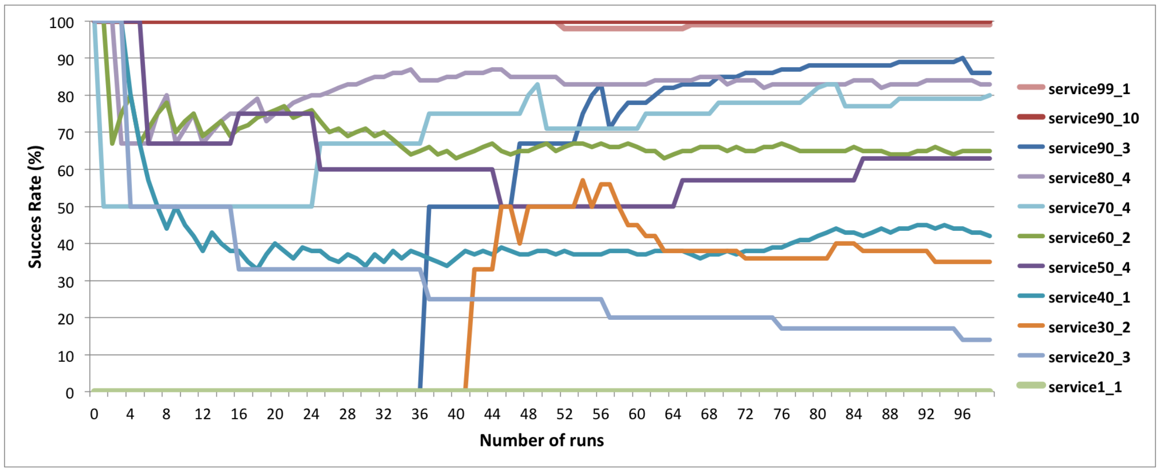


Figure 13: Service Precision Evaluation with Knapsack Selection

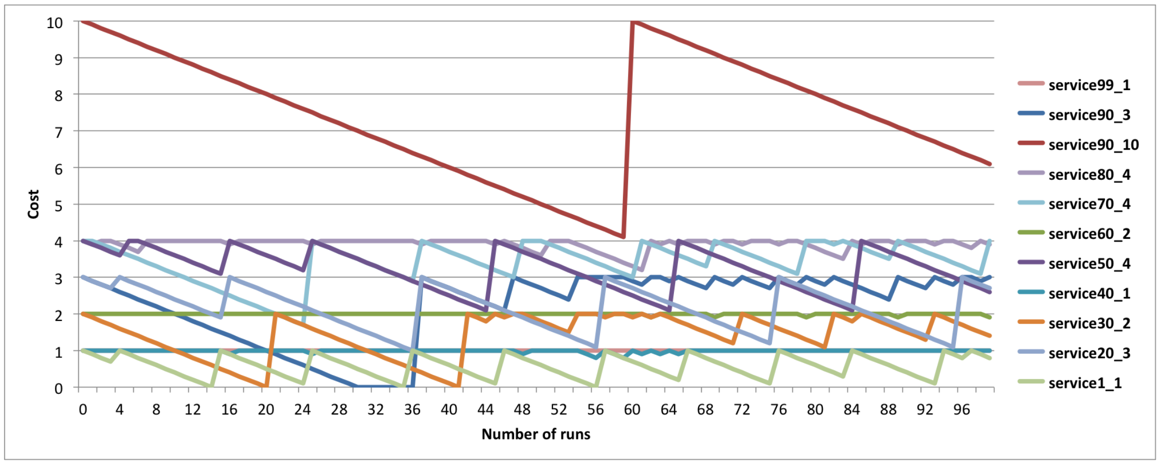


Figure 14: Service Cost Evaluation with Knapsack Selection

our work. As we decrease unselected services' costs by *0.1* after each run, knapsack algorithm eventually selects them and based on their success in returning expected result we update their precision average. We also update their cost values based on service's run time. The changes in services' costs are represented in Figure 14 and the select counts of services are represented in Table 10. The results shows that we didn't completely ignore high cost services while low cost services are used more frequently.

On the other hand, Figure 15 represents success rate of the system in policy level, using 0.5 probability as policy hold rate. When static cost and precision values are

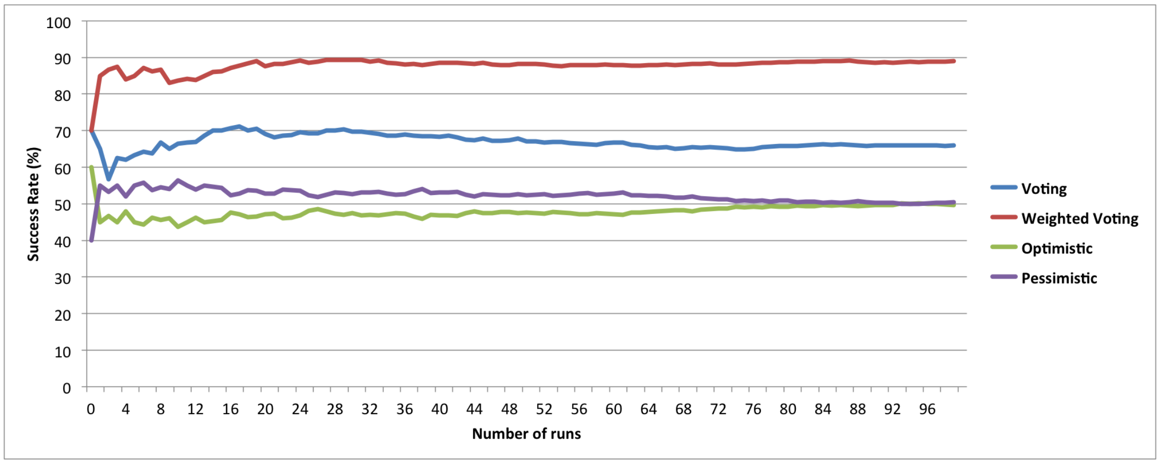


Figure 15: Policy Success Rates with Service Cost-Precision Evaluation

used, as mentioned above, we observed that some services are never used because of their initial values. When cost and precision values are subject to change, this test achieves to balance exploration and exploitation by selecting low-cost high-precision services more frequently than high-cost low-precision services and concluding with similar success rate to the success rate in Figure 10.

Table 10: Service Select Count

Service	*Count*	*Success*	*Fail*
service99_1	100	99	1
service90_3	22	19	3
service90_10	2	2	0
service80_4	64	53	11
service70_4	15	12	3
service60_2	95	62	33
service50_4	8	5	3
service40_1	90	38	52
service30_2	17	6	11
service20_3	7	1	6
service1_1	12	0	12

CHAPTER X

CONCLUSION

In this work, we discussed the necessity of information gathering while reasoning system with incomplete knowledge. Incomplete information may lead to wrong inferences and possible policy violations. By introducing information gathering system to the reasoning process, we improve the efficiency of the policy compliance.

Our approach consists of using an intelligent information gathering system to improve decision making process. To accomplish this, we focused on open-world reasoning and abduction to get useful explanations for possible policy violations. Instead of relying on initial data and inferring a conclusion, we utilise external services to complete our knowledge base based on the explanations provided by abduction algorithm. We used a Prolog engine to seamlessly integrate abduction and information gathering system. External information services are stored in HyperCat catalogue server because of it's simplicity and extendability with custom properties. We discussed the service selection using Knapsack algorithm in order to select most efficient services. With the information provided by services, our framework will decide whether a policy is violated.

We introduced real world examples using our framework. We gathered information to form our knowledge base and created policies based on the needs of the systems. Using abductive reasoning, we found the missing information that needs to be known in order to make best decision. We gathered the missing information from sample external services that are registered to our catalogue server. After we completed our knowledge base, we found the violated policies and took actions accordingly.

We compared the performance of different service combination approaches and we introduced a validation process for service cost and precision values. We showed that the *weighted voting* system is more efficient at selecting the optimum results while considering the variability of precision and cost values of services.

Bibliography

[1] R. Lea, *HyperCat: an IoT interoperability specification.* IoT ecosystem demonstrator interoperability working group, 2013.

[2] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok, "Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder", *Lecture Notes in Computer Science*, vol. 2870, pp. 419–437, 2003.

[3] H. Zhao, J. Lobo, and S. M. Bellovin, "An algebra for integration and analysis of ponder2 policies", in *POLICY '08: Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, (Washington, DC, USA), pp. 74–77, IEEE Computer Society, 2008.

[4] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment", in *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, pp. 63–74, 2003.

[5] A. Uszok, J. M. Bradshaw, J. Lott, M. Breedy, L. Bunch, P. Feltovich, M. Johnson, and H. Jung, "New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAoS", in *POLICY '08: Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, pp. 145–152, 2008.

[6] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao, "Understanding and capturing people's privacy policies in a mobile social networking application", *Personal Ubiquitous Comput.*, vol. 13, pp. 401–412, Aug. 2009.

[7] P. Klemperer, Y. Liang, M. Mazurek, M. Sleeper, B. Ur, L. Bauer, L. F. Cranor, N. Gupta, and M. Reiter, "Tag, you can see it!: Using tags for access control in photo sharing", in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 377–386, 2012.

[8] D. Poole, "Representing bayesian networks within probabilistic horn abduction", in *In Proc. Seventh Conf. on Uncertainty in Artificial Intelligence*, pp. 271–278, 1991.

[9] T. Sato, "A statistical learning method for logic programs with distribution semantics", in *ICLP 1995*, pp. 715–729, 1995.

[10] N. Zhou, T. Sato, and K. Hasidad, "Toward a high-performance system for symbolic and statistical modeling", in *Working Notes of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data (SRL-2003)* (L. Getoor and D. Jensen, eds.), (Acapulco, Mexico), pp. 153–159, August 11, 2003.

[11] P. Flach, A. Kakas, and O. Ray, "Abduction, induction, and the logic of scientific knowledge development", in *Proceedings of the Workshop on Abduction and Induction in AI and Scientific Modelling*, 2006.

[12] A. C. Kakas and F. Riguzzi, "Abductive concept learning", in *New Generation Computing*, pp. 243–294, 1999.

[13] S. Muggleton, F. Mizoguchi, and K. Furukawa, "Special issue on inductive logic programming", *New Generation Computing*, 1995.

[14] O. Ray, *Hybrid abductive-inductive learning.* PhD thesis, Department of Computing, Imperial College London, 2005.

[15] S. MUGGLETON, "Inverse entailment and progol." `http://www.doc.ic.ac.uk/~shm/Papers/InvEnt.pdf`, 1995.

[16] "UDDI Technical White Paper." `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`, 2000.

[17] E. WebSphere, "Enabling soa using websphere messaging", 2006.

[18] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study", in *Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering, CAINE*, 2009.

[19] A. Jøsang, "Conditional reasoning with subjective logic", *Journal of Multiple-Valued Logic and Soft Computing*, vol. 15, no. 1, pp. 5–38, 2009.

[20] E. Denti, A. Omicini, and A. Ricci, "Multi-paradigm Java-Prolog integration in sf tuProlog", *Science of Computer Programming*, vol. 57, pp. 217–250, Aug. 2005.

[21] P. Mancarella, G. Terreni, F. Sadri, F. Toni, and U. Endriss, "The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments", *Theory Pract. Log. Program.*, vol. 9, pp. 691–750, 2010.

[22] A. C. Kakas, B. Van Nuffelen, and M. Denecker, "A-system: problem solving through abduction", in *Proceedings of the 17th international joint conference on Artificial intelligence (IJCAI'01)*, pp. 591–596, 2001.

[23] J. Du, G. Qi, Y.-D. Shen, and J. Z. Pan, "Towards practical abox abduction in large owl dl ontologies", in *The Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)*, (San Francisco, USA), August 2011.

[24] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations.* New York, NY, USA: John Wiley & Sons, Inc., 1990.

[25] "Guide to allowing and blocking ads on your site." `https://support.google.com/adsense/answer/180609?hl=en&ref_topic=23390&rd=1`.

[26] Prof. Dr. Ferhunde Öktem, Doç. Dr. Melike Sayıl, Dr. Sevilay Çelenk Özen, "Akıllı işaretler sınıflandırma sistemi." http://www.rtukisaretler.gov.tr/AIAdministration/download/akademikRapor_3.doc.

[27] "tuprolog guide." http://tuprolog.sourceforge.net/doc/2p-guide.pdf.

[28] T. Stafford, "Fundamentals of learning: the exploration-exploitation trade-off." http://www.tomstafford.staff.shef.ac.uk/?p=48.

[29] J. Ma, A. Russo, K. Broda, and E. Lupu, "Distributed abductive reasoning with constraints", in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, (Richland, SC), pp. 1381–1382, International Foundation for Autonomous Agents and Multiagent Systems, 2010.