From Truth Maintenance to Trust Management on the Semantic Web *

Lalana Kagal[1], Ian Jacobi[1], and Ankesh Khandelwal[2]

[1] MIT CSAIL
Cambridge, MA 02139
{lkagal, jacobi}@csail.mit.edu
[2] Rensselaer Polytechnic Institute
Troy, NY 12180
ankesh@cs.rpi.edu

Abstract. The Semantic Web is a decentralized forum on which anyone can publish their structured data or extend and reuse existing data. This inherent openness of the Web raises questions about the trustworthiness of Web data — which data sources to trust, what data of these sources to trust, how to make declarations about trust, how to use these declarations to calculate trust, etc. This paper does not propose a new trust management model or mechanism. Instead, it shows how AIR, a Web rule language, can be used to develop different kinds of trust models to address trust issues of the open Web. The AIR rule language is based on Semantic Web technologies and uses *truth maintenance* to track how inferences are deduced. It has several useful features that can be used to model trust of data sources, data, and rules as well as inferences made by rules. It also reasons over this information to make further inferences. AIR generates justifications for all inferences made, allowing the trustworthiness of inferences to be evaluated by consumers of this data. In this paper, we provide an overview of AIR and illustrate how it can be used for trust management using an example from the financial domain.

1 Introduction

The Semantic Web provides technologies for rendering machine-readable data on the Web. These technologies include the Resource Description Framework (RDF) [3], a standard data model, RDF Schema (RDFS) [6], a language for describing vocabularies such as class and property hierarchies using RDF, and the Web Ontology Language (OWL 1 & 2) [2, 1], an ontology language for expressing all kinds of domain knowledge, including complex relationships between classes and properties. Much of this data is generated and maintained by multiple individuals or organizations in a decentralized manner over the Web, with incentives for reuse. This form of data, also known as *Linked Data* [4], is interconnected but spatially dispersed. Rules must be able to dynamically traverse this web of

* This work was supported in part by NSF Cybertrust award number 04281, IARPA award number FA8750-07-2-0031, and AFOSR award number 6921570.

data to obtain additional facts, if required, to support conclusions. Given the openness of the Web, however, the reliability of data depends on trustworthiness of the source, the maintainer of the data, the last date it was updated, etc. For example, there might be multiple Friend Of a Friend (FOAF)[3] files for Tim Berners-Lee that describe his social profile in RDF, however, the one that is most trusted is the one available at the W3C website. This is due to the trustworthiness of the source, W3C. Things that are deduced from data sources also have derived trustworthiness from those of the sources and of any rules used in the deduction. In this work, we consider different aspects of trust-based usage of Web data including associating trust with rule-based deductions.

AIR (**A**ccountability **I**n **R**DF) is a Semantic Web-based rule language. It provides a version of production rules that can exist on separate Web servers but can be linked together, also called nested rules, allowing for modular development [13]. This modularity and nested activation reflect how rules in laws, security policies, business rules and work-flows are defined. AIR is represented in N3 [3], a human-readable representation of the RDF data model, and supports a rich set of functions for selectively accessing trusted content from the Web as well as cryptographic, string, and math operations. Furthermore, AIR supports contextually scoped reasoning, i.e. rule conditions can be scoped to be satisfied against different datasets or against deductions from different rules and datasets. Note that these datasets and rules can be specified in a rule definition, and can be dynamically selected, such as through the use of trust metrics.

The AIR reasoner provides detailed justification (explanations) for deductions by using a *Truth Maintenance System*, which keeps track of the logical structure of a derivation. The justification gives the details of the rules that fired, the order in which they fired, and the data sources that contributed to the satisfaction of a rule condition. This information is particularly useful when evaluating the trustworthiness of an inference.

We may have different degrees of belief for rules and data and may specify trust values that represent those degrees. These trust values can be used when the rules are defined and may be incorporated in the rule condition such that only those inferences that can be trusted are deduced. These deductions can be further filtered by the rules that use them based on new beliefs. In the latter case, information on the rules that fired and the datasets used to make the deduction may be obtained from the justification produced by the AIR reasoner.

This paper is structured as follows: we begin by introducing existing work dealing with trust on the web. Then, in section 3, we discuss trust problems on the web by giving an example. Section 4 gives an overview of the AIR language and how reasoning is performed. In section 5 we describe how AIR can be used for trust management on the Web. Finally, section 6 provides a summary and directions for future work.

[3] http://www.foaf-project.org/

2 Related Work

Well-known trust management systems such as PolicyMaker [16], KeyNote [5], REFEREE [7], and Delegation Logics [15] view trust management as an authorization problem. They define mechanisms for inferring whether a requester (software or human agent) is permitted to perform a certain action or access a certain resource based on a set of constraints defined by the action/data owner. PolicyMaker, one of the first distributed trust management schemes proposed, reasons over policies, a set of credentials, and a string describing the secure action and can reply with a yes/no answer dependent on whether the credentials meet/do not meet the requirements of the policy. It may also provide additional requirements that would make the request acceptable. KeyNote is the next generation of PolicyMaker. The basic function of both PolicyMaker and KeyNote is to be able to answer queries about authorization; neither is aimed at handling trust issues of data or data sources. REFEREE was designed to facilitate security decisions for Web browsing. It is similar to PolicyMaker as it allows its assertions (credentials and policies) to be described in a programming language. Unlike PolicyMaker, however, its trust management engine is able to fetch additional credentials while evaluating a request and can also perform cryptographic-signature verification. Delegation Logic (DL) is a logic-based language for authorization in distributed systems and is used to represent authorization through delegations.

Our approach is different from the above approaches in that it is focused on evaluating the trustworthiness of data on the open Web and on allowing decision making mechanisms about trust to be declaratively specified. Approaches most related to ours are those which discuss how trust values for users and data sources can computed such as [18, 10, 11, 14]. Richardson et al. enable users to maintain trust for other users and provide functions to merge these values into trust values for all users by leveraging the path of trust between users [18]. Kuter et al. allow users to maintain trust values or trust estimates for data sources and provide a probabilistic technique to use that information to compute a trust estimate for a data source [14]. Our approach can be thought of as a meta-modeling approach that allows different trust frameworks to be declaratively developed and possibly combined. It provides a rule language, mechanisms for accessing the Web and cryptographic, math, string and other related functions for specifying how trust is assigned and calculated.

3 Trust Problems on the Web

There are two levels of trust we must have before we can establish a complete determination of trust in Web reasoning: trust in data sources and trust in rules. Together, these two levels of trust may be combined to synthesize a level of trust in the inferences made by a reasoner. Trust in data sources forms the cornerstone of trust on the Web, as no trust may be established without a firm foundation in the data we utilize in making decisions. It is worthwhile to consider rules in which the trust of a data source itself plays a role in determining whether a rule may have matched.

Consider the following example: a stock broker, Big Bucks, Inc., makes buy and sell recommendations to its clients based on an algorithm that looks at various news, blog, and discussion forum feeds and then matches them against desired criteria for each client. Isabel, a client of Big Bucks, has a fairly aggressive portfolio and is willing to purchase any company for which a recommendation is published. Unfortunately, not every stock analyst is trustworthy, as they may have a vested interest in seeing certain stocks outperform others. As a result, Isabel may require a high level of trust in the sources in which the recommendations are published.

More complex policies must also be represented. Another customer of Big Bucks, Jay, is more conservative than Isabel, and may requires two buy recommendations from stock analysts (although he trusts their sell recommendations more) *and* requires that the stock be above its 200 day moving average. Since Jay trusts some statements more than other statements, we must be able to differentially select trust values dependent on the type of data used from a source.

The second level of trust we must consider is that of trust in rules. Although being able to trust base facts is an important factor in reasoning, many scenarios may have some level of trust integrated into the rules themselves. Rules may be probabilistic or produce only partially trustworthy results due to a lack of certainty about the rule itself.

For example, some of Big Bucks's customers may rely on rules that Big Bucks itself has made available for use for its customers. Karl uses Big Bucks's rules as part of his decision making process, but does not entirely trust them, as Big Bucks may have a conflict of interest in making suggestions to its customers. Karl may wish to only partially trust Big Bucks's rules, depending on the output of other rules to corroborate Big Bucks's conclusions as to what stocks are worth buying or selling.

A framework that is flexible enough to capture these requirements would be useful for trust management for Web data. In the following sections, we show how AIR, a general Web rule language, can be used to express different kinds of trust in data, data sources, and rules and combine them in various ways to compute trust values for Web data.

```
@prefix s: <http://s.example.org/ontology#>.
@prefix b: <http://b.example.org/ont#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://src1.example.org/JohnAnalyst#>.

@forSome :X .

:X a <http://www.example.com/shoe>.
s:AmazonFootwear s:rec s:Buy.
a:BestPhone s:rec s:Buy.
a:ConArtist s:rec s:DontBuy.
:Weather :prediction :VeryHot.
b:SARS rdf:type b:PotentialVirusOutbreak.
```

Fig. 1. Content of an RDF Data Source

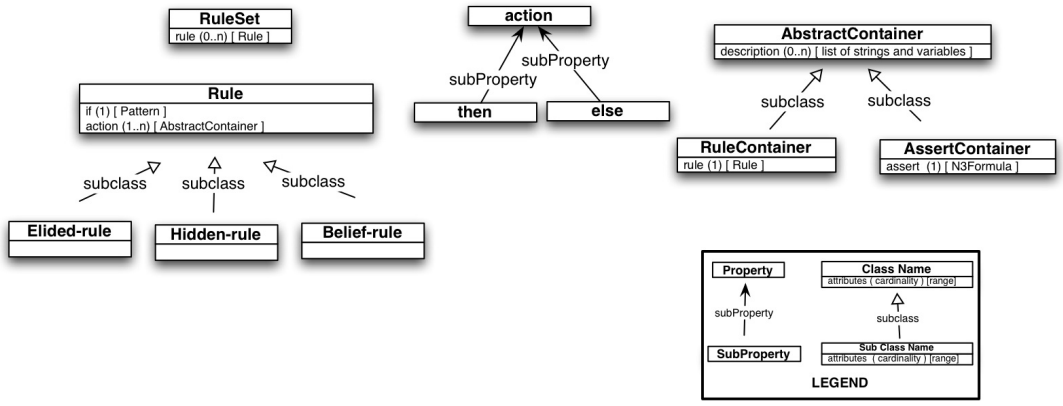

Fig. 2. AIR Rule Ontology

4 Overview of AIR

The rule language we use, AIR, is written in N3 [3], a syntax for representing Semantic Web data based on the RDF abstract syntax. N3 makes use of a number of basic concepts from RDF, including the concept of triples, which correspond to the logic predicate *holds(subject, predicate, object)*. The logical terms *subject*, *predicate*, and *object* may be literals, existentially quantified resources known as "blank nodes", or opaque Uniform Resource Identifiers (URIs) denoted inside angle brackets or using qualified names (QNames)[4]. For example, in Figure 1, the triple, ***b:SARS rdf:type b:PotentialVirusOutbreak*** has *b:SARS* as *subject*, *rdf:type* as predicate and *b:PotentialVirusOutbreak* as *object*. N3 extends RDF's abstract syntax by adding formula quoting and universal quantification. Figure 1 provides some examples of N3 triples. RDF documents consist of conjunctions of these triples which may also be called graphs or formulas. Please refer to `http://www.w3.org/2000/10/swap/Primer` for an overview of N3 and to the Appendix for the list of namespaces used in the paper.

AIR is made up of a set of built-in functions and two independent ontologies — the first is for the specification of AIR rules, and the second deals with describing justifications for the inferences made by AIR rules. The built-in functions allow rules to access Web resources, query SPARQL Query Language for RDF (SPARQL) endpoints [17], and perform scoped contextualized reasoning, as well as basic math, string and cryptographic operations. While developing the rule vocabulary, we focused on capturing how real world rules and laws are written to allow them to be represented naturally in AIR. For the justification vocabulary, our focus was on re-usability of justifications and on automated proof checking. When given as input some AIR rules, defined in the AIR rules vocabulary, and some Semantic Web data, the AIR reasoner produces a set of inferences that

[4] http://www.w3.org/TR/REC-xml-names/#ns-qualnames

are annotated with justifications, described in the justification vocabulary. The runtime input to AIR rules can be any RDF graph or an empty graph, if the rules only access Web resources. In the following subsections we cover the AIR rule ontology and built-in functions; the AIR reasoner and representation of AIR justifications.

4.1 AIR Rules

The AIR rule vocabulary consists of several key classes and properties, shown in Figure 2, which are identified by their QNames. ***air:Belief-rule*** is a class of resources representing the set of all rules for which full justifications are made. These rules may then have the properties ***air:if***, ***air:then***, and ***air:else*** associated with them to represent the N3 pattern to be matched, and the actions to take if the pattern matches, or does not match, respectively. ***air:then*** and ***air:else*** actions may be described in terms of the facts they assert (using the ***air:assert*** property) or the rules they cause to match next (using the ***air:rule*** property). Figure 3 demonstrates how AIR rule vocabulary can be used to define Isabel's simple rule about recommending stocks. The rule suggests that Isabel only buy stock in a certain company, ***air:assert { :Isabel s:shouldBuy :COMPANY}***, if a data source such as a stock analyst of a trust value greater than or equal to 5, ***:TRUST1 math:notLessThan 5***, recommends that the stock of that company be bought, ***:COMPANY s:rec s:Buy*** .

```
@forAll :SRC1,:TRUST1.

:IsabelBuyRule a air:Belief-rule ;
    air:if {
        :SRC1 log:includes { :COMPANY s:rec s:Buy . }.
        :SRC1 t:trustvalue :TRUST1.
        :TRUST1 math:notLessThan 5. }
    air:then [ air:assert { :Isabel s:shouldBuy :COMPANY . } ] .
```

Fig. 3. Example AIR Rule: `IsabelBuyRule` uses the trust ontology described in Figure 5.b to assign trust to data sources. Isabel only buys stock in a certain company if a data source such as a stock analyst of a trust value greater than 5 recommends that the stock of that company be bought

AIR supports several functions including cryptographic, math, string, list and time functions. ***:TRUST1 math:notLessThan 5*** is an example graph pattern that uses the `math:notLessThan` built-in function. The subgraph of ***:TRUST1 math:notLessThan 5*** matches if the value of `:TRUST1`, which is a variable, is found to be greater than or equal to 5. Similarly, ***:KEY crypto:md5 :HASH*** calculates the MD5 hash of `:KEY` and sets `:HASH` to it. AIR also includes functions for signing and verifying signatures and certificates.

Traditionally, rule languages have been designed with the assumption that all rules can use all the input data for deductions. However, this isn't very desirable

in a Web rule language, especially for trust-based deductions. As we will also see through our examples, different rules may require restricting of scope of applicability to a subset of input data or documents, based off belief in the data or documents. Sometimes the data is not just defined existentially but also has an intentional component, defined through rules (which we may call int-def-rules). In order to access the complete data we must be able to access the extensional as well as intentionally-defined data, without risking unintended application of int-def-rules to the entire input data. Note that the two scenarios described above are different aspects of contextually scoped reasoning. AIR includes functions that are useful in manipulating remote data, and that support contextually scoped reasoning such as ***log:semantics***, ***log:includes*** and ***air:justifies*** built-in functions.

The ***log:semantics*** built-in fetches a document and returns its representation as an N3 graph. This N3 graph may then be used in conjunction with the ***log:includes*** and ***log:notIncludes*** built-ins to determine whether a subgraph is, or is not, present in the graph. These ***log*** built-ins can be used to selectively access content from Web resources. ***air:justifies*** is a particularly powerful built-in function that executes some external AIR rules against some Semantic Web data and checks whether they produces a certain RDF graph. It returns a graph containing both the inferred results as well as the justification of the facts. In Figure 6, ***air:justifies*** is used to execute a set of rules and the inferred recommendation is only accepted if the inference was made by a trusted rule. SPARQL queries can be executed from within AIR rules using ***sparql*** built-ins. SPARQL CONSTRUCT queries may be sent to SPARQL endpoints using the ***sparql:queryEndpoint*** property assertion, and subgraph patterns may be searched for in the graph returned by the endpoint.

4.2 AIR Reasoner

A deductive reasoning system derives conclusions from previous deductions or premises by the application of deductive rules. For any given conclusion, it is useful to know the specific set of premises that it was derived from; this set is called the set of dependencies for the conclusion. Dependency tracking is the process of maintaining dependency sets for derived conclusions.

Some dependency-tracking mechanisms provide additional features. For example, a Truth Maintenance System (TMS) keeps track of the logical structure of a derivation, which is an effective explanation of the corresponding conclusion. Another useful feature, also provided by a TMS, is the ability to assume and retract hypothetical premises. There are several reasons why dependency tracking is useful for trust management systems: (i) the dependency set for a result provides a natural focus when trying to solve policy compliance problems, and (ii) it can provide a concise explanation for a result. This is essential for confirming that a trust framework is correctly modeled. It can also help identify situations where a framework is having unanticipated or undesirable consequences.

AIR uses a *Truth Maintenance System* as the dependency tracking mechanism [12]. The TMS provides considerable power in a very simple mechanism;

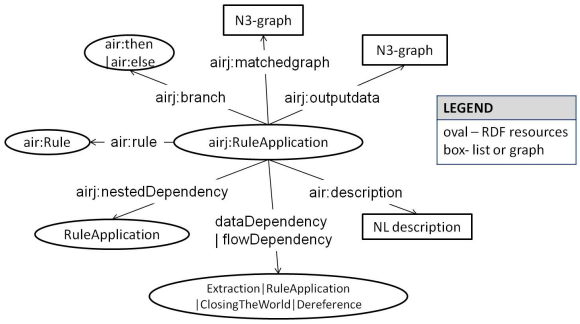

Fig. 4. Ontology for description of a rule firing event

its primary cost is the memory required to record the structure of a derivation. Although the TMS technology was invented in the 1970s [8], and recent work in applying truth maintenance systems to problem solving was done in the 90s [9], it is not well known outside the artificial intelligence community.

Our reasoner is a deductive reasoning system, where condition of a rule, which is production-rule like, is a pattern to be matched against a set of believed statements. When the pattern matches, or it fails to match, the rule's action is performed. Typically the action asserts new beliefs, causing them to be added to the set of believed statements. When something is added to the belief set, it is associated with a justification for its belief. In the case of a simple statement of fact, there is a trivial justification that the statement is an assumption. A derived statement has a justification based on the inputs used to make the derivation.

In such a simple rule system, all of the dependency information is implied by the rules themselves. If a new belief is asserted by a rule's action, then its justification is the set of statements that matched the rule's pattern. More precisely, we believe the asserted statement if and only if we believe every one of the matched statements. Additionally, the justification records an identifier for the rule; this identifier together with the matched statements provide all the relevant information about the particular deduction step just performed. Typically these deduction steps build on one another, resulting in a tree-like justification structure for any given belief, in which the belief is the trunk of the tree and the assumptions are the leaves. This tree structure is a complete explanation of the support for the belief.

The afore-mentioned dependencies are captured through an ***Event*** based ontology. There is an event for every important operation performed during reasoning - ***Dereference***, ***BuiltinAssertion***, ***BuiltinExtraction***, ***RuleApplication*** and ***ClosingTheWorld***.

Dereference is used to relate the documents, of both data and rules, with their graph representations. The built-in triples are premises and they are computed as needed by the reasoner. The output data of the abstract event ***BuiltinAssertion*** is assumed to be all the triples that hold for a particular built-in.

Ontology Type	Ontology Definition	Example Assignments
a. Simple binary trust	TrustedSource rdf:type rdfs:Class. UnTrustedSource rdf:type rdfs:Class	<http://src1.example.org> rdf:type TrustedSource. <http://srcx.example.org> rdf:type UnTrustedSource
b. Trust value	trustvalue rdf:type rdf:Property.	<http://src2.example.org> trustvalue 8. <http://src1.example.org> trustvalue 2
c. Qualified trust	trust rdf:type rdf:Property; range TrustVal. tval rdf:type rdf:Property; domain TrustVal. tpattern rdf:type rdf:Property; domain TrustVal.	<http://src1.example.org> trust [tval 8; tpattern { :VARIABLE s:rec s:Buy}]. <http://hospital.example.org> trust [tval 99; tpattern { :VAR1 rdf:type h:PotentialVirusOutbreak; h:duration :VAR2 }]
d. Trust associated with rules	trustvalue rdf:type rdf:Property	<http://bigbucks.example.com/rules #BuyRule> trustvalue 7

Fig. 5. Example Trust Declarations for Data Sources and Rules

The specific triples that were computed are represented as ***outputdata*** of the ***BuiltinExtraction*** event, which refers to the abstract operation of extracting the triple from all the facts.

Firing of a rule is captured as a ***RuleApplication*** event. Figure 4 describes various properties associated with a ***RuleApplication*** event. The rule-id is given by the ***air:rule*** property. Whether the ***then*** or ***else*** action fired is described using ***airj:branch***. If the ***then*** branch fired then the subset of data that satisfied the condition is declared using ***airj:matchedgraph***. The ***else*** actions fire only if no ***then*** actions can be fired for all other rules. Before the ***else*** actions actually fire, the world is closed (temporarily) assuming all the failed rule conditions to be false. The closing of world is captured using the ***ClosingTheWorld*** event. When a nested rule fires a ***nestedDependency*** is established with the firing of the parent rule event, which activated the nested rule. The triples asserted when the rule fired, are associated with the ***air:outputdata*** property.

In Figure 7 we filter those buy recommendations for Karl, ***:Karl s:shouldBuy :COMPANY***, that are supported by a trusted `:RULE` defined in `http://bigbucks.example.com/rules`. The rule searches the justification returned by AIR reasoner for a ***RuleApplication*** event, `:RULEAPPEVENT`, such that some `:RULE` fired and asserted the buy recommendation.

5 AIR and Trust Management

As AIR is a Semantic Web-based rule language it is able to reason over different semantic representations for trust. The language does not insist on any specific representation of trust and allows the user/system to provide a representation that best captures their requirements. Figure 5 provides some examples of how trust could be assigned to data sources. Figure 5.a is a simple way of representing binary trust. It defines 2 classes, `TrustedSource` and `UnTrustedSource`, and a

```
@forAll :SRC1, :SRC2, :TRUST1, :TRUST2, :COMPANY, :PRICE, :MA200.

:JayBuyRule a air:Belief-rule ;
    air:if {
        :SRC1 log:includes { :COMPANY s:rec s:Buy . } ;
              t:trust [ t:pattern { :VARIABLE s:rec s:Buy . } ;
                                t:value :TRUST1 ] .
        :SRC2 log:includes { :COMPANY s:rec s:Buy . } ;
              t:trust [ t:pattern { :VARIABLE s:rec s:Buy . } ;
                                t:value :TRUST2 ] .
        :TRUST1 math:notLessThan 7 .
        :TRUST2 math:notLessThan 7 .
        <http://stocks.example.com/ticker> log:includes {
              :COMPANY s:stockPrice :PRICE ;
                  s:TwoHundredDayMA :MA200 . } .
        :PRICE math:greaterThan :MA200 .
    } ;
    air:then [ air:rule :CompareSources ] .

:CompareSources a air:Belief-rule ;
    air:if { :SRC1 owl:sameAs :SRC2 . } ;
    air:else [ air:assert { :Jay s:shouldBuy :COMPANY . } ] .
```

Fig. 6. Compute trust in data using trust in data sources: `JayBuyRule` uses the trust ontology described in Figure 5.c to assign trust to data sources with respect to buy recommendations. If there are two different data sources with trust values greater than or equal to 7 that recommend the same stock, and if the stock is above its 200 day moving average, then the rule recommends that Jay should buy that stock.

source can be declared to be either trusted or untrusted depending on which class it belongs to. Figure 5.b illustrates how trust values can be assigned to sources through a `trustvalue` property. Figure 5.c is an example of trusting a source with respect to certain data. For example, a hospital site may be trusted with information about a potential virus outbreak but may not be trusted with respect to its inflation predictions. The `tpattern` is a graph pattern property and it signifies that only RDF data from the source that match the pattern will be trusted. For example, if the content of a trusted site is as shown in Figure 1 and the `tpattern` is in Figure 5.c, then only the triples that match the `tpattern`, ***s:AmazonFootwear s:rec s:Buy. a:BestPhone s:rec s:Buy.*** , will be trusted and the triples regarding the virus outbreak and weather will be ignored. AIR currently supports simple graph patterns to specify the kinds of data trusted from a certain source. Similar representations are possible for rules, for example, Figure 5.d shows a trust value associated with a AIR rule, `BuyRule`.

AIR rules can be written to consume these trust declarations and combine them in different ways in order to compute trust values for data or inferences of interest. `JayBuyRule` uses the trust ontology described in Figure 5.c to assign trust to data sources with respect to buy recommendations. `:JayBuyRule` recommends that Jay buy a stock if there are two different data sources with trust values greater than 7 that recommend it and if the stock is above its 200 day moving average. The rule could be modified to calculate the minimum or average trust of all sources recommending a particular stock and estimate how

```
@forAll :RULESET, :DATA, :RULEJUST, :COMPANY, :RULE, :TRUST.

:KarlBuyRule a air:Belief-rule ;
    air:if {
        <http://bigbucks.example.com/rules> log:semantics :RULESET .
        ((:RULESET) (:DATA)) air:justifies :RULEJUST .
        :RULEJUST log:includes {
            @forSome :RULEAPPEVENT .
            :Karl s:shouldBuy :COMPANY .
            :RULEAPPEVENT pmlj:outputdata { :Karl s:shouldBuy :COMPANY . } .
            :RULEAPPEVENT air:rule :RULE .
        } .
        :RULE t:trust :TRUST .
        :TRUST math:notLessThan 7 .
    } ;
    air:then [ air:assert { :Karl s:shouldBuy :COMPANY . } ] .
```

Fig. 7. Compute trust in data using trust in rules: `KarlBuyRule` uses the trust ontology described in Figure 5.d to assign trust to the rules used by Big Bucks to generate buy recommendations. If the rule used to generate a buy recommendation has a trust value greater than or equal to 7, then the rule recommends that Karl should buy that stock.

trusted the recommendation to buy the stock is. Probabilities associated with data sources could also be handled similarly.

AIR rules can also reuse and execute other AIR rules. As rules themselves could have trust values associated with them, as in Figure 5.d, it is possible to reason about the trustworthiness of rules and deduce trust of inferences made by them. AIR's support for the ***air:justifies*** property allows for the execution of other rules which we may be able to query for trust. The rule `:KarlBuyRule` in Figure 7 encapsulates such a rule and recommends that Karl buy a stock only if the rule that is used to justify the recommendation is trusted by Karl with a trust value greater than or equal to 7.

In `:KarlBuyRule`, ***air:justifies*** is used to run the rules at the URL <http://bigbucks.example.com/rules> (as extracted with ***log:semantics***) against some trusted data. The result of this reasoning is stored in the output variable `:RULEJUST`, which may be used with other built-in functions, like ***log:includes***, to determine not only which facts are asserted by the rules, but also the justifications for such. These justifications may then be used to determine the rules which caused some conclusion to be found to be true and their trust values.

6 Summary and Future Work

In this paper, we discussed how AIR, a Semantic Web-based rule language, could be used to model and reason over trust of data sources, rules and inferences. We discussed several ways of modeling trust including binary trust, simple trust values and qualified trust for both data sources and rules and showed how these trust values could be used and combined by rules for trust management. Though this work demonstrates the usefulness of AIR, it relies on user-generated rules

for handling trust. As part of our future work, we will work on general rules that will handle trust transparently such that users do not need to explicitly know about or handle trust in their systems but will be able to customize these rules to do it for them. These rules will also take other factors into consideration while computing trust, such as the maintainer of the data, the last date it was updated, and frequency of updates, which could either be meta-data associated with the data itself or available from public annotation services such as Annotea [19].

References

1. OWL 2 Web Ontology Language, W3C Recommendation 27 October 2009. `http://www.w3.org/TR/owl2-overview/`, 2009.
2. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004. `http://www.w3.org/TR/owl-ref/`, 2004.
3. T. Berners-Lee. Primer: Getting into RDF and Semantic Web using N3. http://www.w3.org/2000/10/swap/Primer, 2005.
4. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.
5. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System Version. Internet RFC 2704, September 1999., 1999.
6. D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation. `http://www.w3.org/TR/rdf-schema`, February 2002.
7. Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for Web Applications. *Computer Networks and ISDN Systems*, 29(8–13):953–964, 1997.
8. J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, November 1979.
9. K. D. Forbus and J. de Kleer. *Building problem solvers*. MIT Press, Cambridge, MA, USA, 1993.
10. Y. Gil and V. Ratnakar. Trusting information sources one citizen at a time. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 162–176, London, UK, 2002. Springer-Verlag.
11. J. Golbeck, B. Parsia, and J. Hendler. Trust networks on the semantic web. In *In Proceedings of Cooperative Intelligent Agents*, pages 238–249, 2003.
12. L. Kagal, C. Hanson, and D. Weitzner. Using dependency tracking to provide explanations for policy management. In *IEEE Policy 2008*, 2008.
13. L. Kagal, I. Jacobi, and A. Khandelwal. Gasping for air: Why we need linked rules and justifications on the semantic web. In *Under review at the International Semantic Web Conference (ISWC)*, 2010.
14. U. Kuter and J. Golbeck. Sunny: a new algorithm for trust inference in social networks using probabilistic confidence models. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 1377–1382. AAAI Press, 2007.
15. N. Li, B. N. Grosof, and J. Feigenbaum. Delegation Logic: A Logic-based Approach to Distributed Authorization. *ACM Transactions on Information Systems Security (TISSEC)*, 6, No. 1, Feb 2003.

16. M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized trust management. *Proceedings of IEEE Conference on Privacy and Security*, 1996.
17. E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf w3c recommendation. http://www.w3.org/TR/rdf-sparql-query, January 2008.
18. M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *Proceedings of the 2nd International Semantic Web Conference*, pages 351–368. Springer Berlin / Heidelberg, 2003.
19. W3C. Annotea project. `http://www.w3.org/2001/Annotea/`.

A APPENDIX: Namespaces

In this paper, we refer to the namespaces defined in Figure 8.

```
@prefix air: <http://dig.csail.mit.edu/2009/AIR/air#>.
@prefix airj: <http://dig.csail.mit.edu/2009/AIR/airjustification#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix sparql: <http://www.w3.org/2000/10/swap/sparqlCwm#>.

@prefix pmlj: <http://inference-web.org/2.0/pml-justification.owl#>.

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

@prefix t: <http://bigbucks.example.com/trust#>.
@prefix s: <http://bigbucks.example.com/stocks#>.
@prefix : <http://bigbucks.example.com/buyPolicy#>.
```

Fig. 8. Namespaces Used