AIR Language Tutorial

March 17, 2009

Ankesh Khandelwal[1], Li Ding[1], Lalana Kagal[2]
[1] Tetherless World Constellation,
Rensselaer Polytechnic Institute, 110 8th St, Troy, NY12180, USA
[2] Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institute of Technology, Cambridge MA 02139, USA
{ankesh,dingl}@cs.RPI.edu, {lkagal}@csail.mit.edu

Abstract. AIR is a Semantic Web rule language that supports customizable explanations for policy decisions. Each AIR policy, represented as a collection of production rules, can be used to check transaction logs that record data manipulation activities. The checking results include (i) asserted policy decision statements, i.e. which certain log entry is "compliant" or "non-compliant" with the given AIR policy; and (ii) customizable justification statements, i.e. the custom-tailored justifications showing how AIR reasoner derived the policy decision statements using the input logs and policies. This paper systematically reviews the language features of AIR with working examples. We also discuss some additional features of a CWM and TMS-based AIR policy reasoner.

Keywords: AIR Language, AIR reasoner, Rule, Policy.

1 Introduction

Accountability in RDF (AIR) [1] is a policy language represented by Turtle [2] with quoting [3]. The current AIR specification [4] uses RDFS [5] based AIR ontology[2] to encode the language constructs and some computational semantics of AIR language. This tutorial complements the AIR specification by (i) elaborating AIR language features with detailed working examples, and (ii) clarifying features associated with AIR reasoner. We also maintain an online version[3] of this tutorial.

The tutorial is structured as follows. Section 2 describes the background data to be used in the examples within the tutorial. Section 3 covers AIR language features with working examples. Section 4 discusses some features specific to AIR reasoner. Section 5 concludes the tutorial with a short discussion.

[1] Please note that the LNCS Editorial assumes that all authors have used the western naming convention, with given names preceding surnames. This determines the structure of the names in the running heads and the author index.
[2] AIR ontology in turtle, http://dig.csail.mit.edu/TAMI/2008/12/AIR/air.ttl
[3] Online version of AIR Policy Tutorial, http://tw.RPI.edu/proj/tami/AIR_Policy_Tutorial

2 Testing Dataset

In Figure 1, we show a simple fictional dataset to be used by the example in the following sections. The dataset is not a transaction log, however, it does carry enough information for testing AIR polices. Following is a simple listing of English sentences corresponding to the RDF statements included the dataset.

- Alice lives in Troy
- Bob lives in Troy
- George lives in Boston
- Alice has NY state ID 307578001
- David has NY state ID 307578002
- Troy is in state of NY
- Boston is in state of MA
- NY is neighbor-state of MA
- neighbor-state is an owl:SymmetricProperty

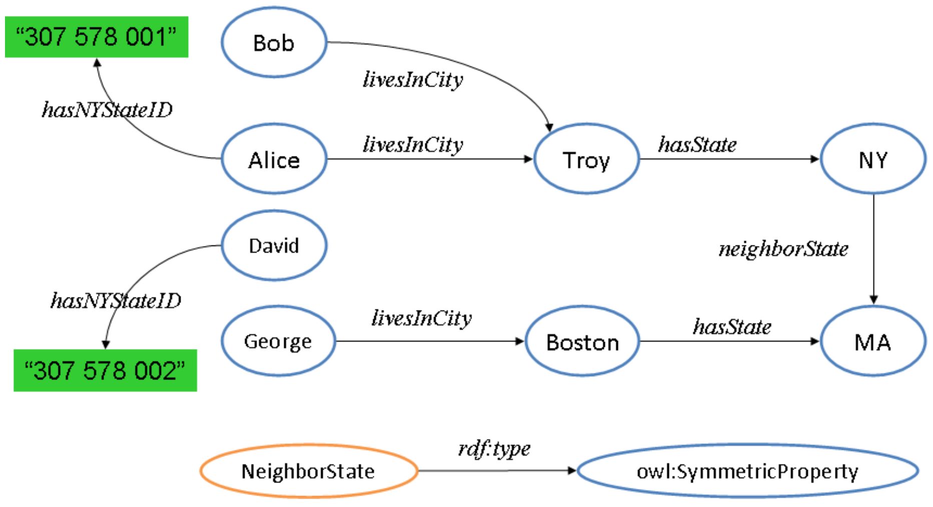

Fig. 1. The RDF graph of the example dataset

The general flavor of policies is that people belonging to NY state- because of their place of residence and/ or if they hold a NY state id- are policy compliant.

3 AIR Language Features

3.1. Policy Modeling

A policy may contain one or more rules. A typical rule contains the following components:

- Variable declaration. The global variables are declared at the beginning of a policy document following N3 syntax.
- Pattern. It is graph pattern of RDF graph for describing the preconditions of the rule. In a graph pattern, a variable may occur at any of the 3 positions, namely subject, object or predicate. It is declared using the air:pattern property.
- Assertion. Some assertions maybe generated as the result of applying the rule. An assertion is declared using air:assert (or air:assertion) property.

3.1.1. Policy with Single Rule

Global variables :PERSON and :CITY are declared first (beginning of the document). :ny_state_residency is a policy that contains the rule :state-residency-rule. :state-residency-rule states that for all person and city when a person lives in that city and the city is in NY state, person complies-with the :ny_state_residency_policy.

Conclusions: Bob and Alice are compliant with :ny_state_residency. (Bob and Alice live in Troy, which is in NY state)

Tutorial Policy 1:

```
@forAll :PERSON, :CITY.
:ny_state_residency_policy a air:Policy;
     rdfs:label "NY State residency policy";
     air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
   rdfs:label "state residency rule";
   air:pattern {
      :PERSON tamip:Lives_in_city :CITY.
      :CITY tamip:Has_state :NY.
   };
air:assert{:PERSON air:compliant-with :ny_state_residency_policy.}.
```

The same policy can be written without explicitly naming the rule. However, for justification of actions taken when this rule is applied the policy engine assigns a random identifier for the rule.

Tutorial Policy 2:

```
@forAll :PERSON, :CITY.
:ny_state_residency_policy a air:Policy;
        rdfs:label "NY State residency policy";
        air:rule [
           rdfs:label "state residency rule";
           air:pattern {
                :PERSON tamip:Lives_in_city :CITY.
                :CITY tamip:Has_state :NY.
           };
           air:assert{:PERSON air:compliant-with :ny_state_residency_policy.}
        ].
```

3.1.1.1: Rule with empty pattern

A rule with an empty pattern can also be asserted as shown in the example below. :troy-rule adds a fact to the data-set that RPI is located in troy. Similarly, :Hartford-rule adds the fact RPI is located in Hartford to the dataset.

Tutorial Policy 21:

```
 :RPI_location_based_policy a air:Policy;
        air:rule :troy-rule;
        air:rule :Hartford-rule.
 :troy-rule a air:Belief-rule;
      rdfs:label "RPI in troy rule";
      air:pattern { };
      air:assert {:RPI tamip:Located_In :troy.}.
 :Hartford-rule a air:Belief-rule;
      rdfs:label "RPI in Hartford rule";
      air:pattern { };
      air:assert {:RPI tamip:Located_In :Hartford.}.
```

3.1.2. Policy with Multiple Rules

Policy can contain multiple rules. Some of these rules are nested, and contained within other rules. All the rules with policy element as their parent are fired. Nested rule is fired only when pattern in the parent rule is matched (or not matched in case of alternative rule triggering), and the variable bindings are passed over to the nested rule.

3.1.2.1. Nested rules

Positively Nested. The following example shows a nested rule becomes active when the pattern gets matched (air:rule).

Policy :ny_state_residency_and_id_policy contains two rules- :state-residency-rule and :state-id-check- where rule :state-id-check is contained in :state-residency-rule. Of-these state-id-check is not active initially. state-id-check rule becomes active only when the pattern in :state-residency-rule is matched, and the variable bindings that occur in state-residency-rule are retained when state-id-check is applied.

According to the rules: for all person, city and ny-state-id, such that the person lives in the city and the city is in NY and the person has a NY state ID as well, the person is compliant with :ny_state_residency_and_id_policy.

Conclusions: Alice is compliant with ":ny_state_residency_and_id_policy.
(Alice lives in Troy which is in NY state and has NY state id 307 578 001)

```
Tutorial Policy 3:
@forAll :PERSON, :CITY, :NY_STATE_ID.
:ny_state_residency_and_id_policy a air:Policy;
        air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
     rdfs:label "state residency rule";
     air:pattern {
          :PERSON tamip:Lives_in_city :CITY.
          :CITY tamip:Has_state :NY.
     };
     air:rule :state-id-check.
:state-id-check a air:Belief-rule;
     rdfs:label "state id check rule";
     air:pattern { :PERSON tamip:Has_ny_state_id :NY_STATE_ID. };
     air:assert {:PERSON air:compliant-with :ny_state_residency_and_id_policy.}.
```

Negatively Nested. The following example shows that nested rule becomes active when the pattern doesn't match (air:alt)

:ny_neighbor_state_residency_policy contains three rules: :non-ny-residency-rule, :neighbor-state-rule and an anonymous rule nested in :non-ny-residency-rule. The anonymous node is positively nested, i.e. this rule is triggered when there is a match for the pattern in :non-ny-residency-rule. In contrast, the :neighbor-state-rule is negatively nested in the anonymous rule. That is, the :neighbor-state-rule would be triggered when the pattern :CITY tamip:Has_state :NY has no match.

According to the :non-ny-residency-rule, if a person lives in a city and if 'the city in NY state' does not hold, apply the :neighbor-state-rule rule. When the :neighbour-state-rule is applied, if the city is in a state, and the state is a neighboring state of NY, the person is asserted to comply with :ny_neighbor_state_residency_policy.

Conclusion: George is compliant with :ny_neighbor_state_residency_policy.
(George lives in Boston, Boston is in MA (not NY), NY is a neighbor state of MA.)

```
Tutorial Policy 4:
@forAll :PERSON, :CITY, :STATE.
:ny_neighbor_state_residency_policy a air:Policy;
     air:rule :non-ny-residency-rule.
:non-ny-residency-rule a air:Belief-rule;
     rdfs:label "Non NY residency rule";
     air:pattern {:PERSON tamip:Lives_in_city :CITY.};
     air:rule [
          air:pattern {:CITY tamip:Has_state :NY.};
          air:alt [air:rule :neighbor-state-rule]
     ].
:neighbor-state-rule a air:Belief-rule;
     rdfs:label "neighbor state rule";
     air:pattern { :CITY tamip:Has_state :STATE.
                         :NY tamip:Neighbor_state :STATE.};
     air:assert { :PERSON air:compliant-with :ny_neighbor_state_residency_policy. }.
```

3.1.2.2. Non-nested rules

The :ny_state_residency_or_id_policy policy contains two rules: :state-residency-rule and :state-id-check, that are not nested and sort of independent of each other. In this case, a person complies with the policy if he has a NY state id or if he stays in a city and that city is in NY. :state-id-check rule checks for ny state id, and :state-residency-rule independently checks if a person lives in a city in NY state.

Conclusions: Alice, Bob and David comply with the :ny_state_residency_or_id_policy. (Bob live in Troy, which is in NY. David has NY state ID 307 578 002. Alice lives in Troy, in NY and has a NY state ID, 307 578 001, as well.)

Tutorial Policy 5:

```
@forAll :PERSON, :CITY, :NY_STATE_ID.
:ny_state_residency_or_id_policy a air:Policy;
        air:rule :state-residency-rule;
        air:rule :state-id-check.
:state-residency-rule a air:Belief-rule;
     rdfs:label "state residency rule";
     air:pattern {
          :PERSON tamip:Lives_in_city :CITY.
          :CITY tamip:Has_state :NY.
     };
     air:assert {:PERSON air:compliant-with :ny_state_residency_or_id_policy.}.
:state-id-check a air:Belief-rule;
     rdfs:label "state id check rule";
     air:pattern { :PERSON tamip:Has_ny_state_id :NY_STATE_ID. };
     air:assert {:PERSON air:compliant-with :ny_state_residency_or_id_policy.}.
```

3.1.3. Alternative Assertion (air:alt)

In Negatively Nested Rules air:alt had been used to activate a nested rule when the pattern in the nesting rule did not match. However, air:alt can also be used to make simply an alternate assertion.

According the the :state-residency-rule: for all persons, person lives in a city and city in state of NY, the person is compliant with :ny_state_residency_policy. And for all persons, person lives in a city and 'city in state of NY' does not hold, the person is not compliant with :ny_state_residency_policy.

Conclusions: Alice and Bob comply with the :ny_state_residency_policy. George is not compliant with the :ny_state_residency_policy. (Alice and Bob live in Troy which is in NY state. George lives in Boston, in MA, which is not in NY.)

Tutorial Policy 16:

```
@forAll :PERSON, :CITY.
:ny_state_residency_policy a air:Policy;
        air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
     rdfs:label "state residency rule";
     air:pattern {
         :PERSON tamip:Lives_in_city :CITY.
     };
     air:rule [
         air:pattern {
             :CITY tamip:Has_state :NY.
         };
         air:assert {:PERSON air:compliant-with :ny_state_residency_policy.};
         air:alt           [air:assert           {:PERSON           air:non-compliant-
 with :ny_state_residency_policy.}]
     ].
```

3.1.4. Variable Quantification and Scoping

- Variables are declared in N3logic syntax and scoped to file in which they appear in.
- Variables can be
    - declared globally (outside policies and rules) or
    - declared in one of the two sides of a rule (pattern or assertion).
- Only globally declared variable can be used on both sides of any rule in that file and can be referred to in nested rules triggered from those rules in that file.
- If one of the rules in the file ultimately triggers a nested rule in a different file, then a previously matched global variable can be used in the latter rule as long as it is referred to with its entire URI (base URI of original file + name of variable).
- A (universal or existential) variable that is declared in the left hand side (pattern) or the right hand side (assertion) of a rule is only accessible in that side.
    - Global declaration of existential variables is not supported in this version. Existentials can only be used in patterns.
- Variables declared in the right hand side (assertion) are simply anonymous nodes with the correct quantification, but that functionality is not supported in this version.

3.1.4.1. Global Declaration

All variables in the examples covered so far were declared globally.

3.1.4.2. Existential Quantification in the pattern

This policy is same as the Tutorial Policy 1 in 3.1, except that :CITY is locally declared within the pattern. Note that :CITY is existentially quantified and is not used in the right hand side (assertion). According to the state-residency-rule, for all persons, person lives in some city that is in NY state, the person is compliant with the :ny-state-residency-policy.

Conclusions: Bob and Alice comply with :ny_state_residency_policy.

Tutorial Policy 6:

```
@forAll :PERSON.
:ny_state_residency_policy a air:Policy;
        air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
     rdfs:label "state residency rule";
     air:pattern {
          @forSome :CITY.
          :PERSON tamip:Lives_in_city :CITY.
          :CITY tamip:Has_state :NY.
     };
     air:assert {:PERSON air:compliant-with :ny_state_residency_policy.}.
```

Instead, if :CITY was universally qualified, the :state-residency-rule would become: for all persons, if person lives in :CITY and for all values that :CITY can take :CITY is in state NY, the person is compliant with :ny_state_residency_policy. Since all URIs in the data are possible values for :CITY, empty set is returned as conclusion. (Tutorial Policy 7, Execute)

Note that the existentially quantified variable :CITY can be replaced with a blank node.

Tutorial Policy 8:

```
@forAll :PERSON.
:ny_state_residency_policy a air:Policy;
        air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
     rdfs:label "state residency rule";
     air:pattern {
          :PERSON tamip:Lives_in_city [tamip:Has_state :NY].
     };
     air:assert {:PERSON air:compliant-with :ny_state_residency_policy.}.
```

3.1.4.3. Universal Quantification in the pattern

TBD

3.1.5. Multiple Policies

3.1.5.1. Multiple policies in one policy document

Here we see that we can define two different policies :ny_state_residency_policy and :ny_state_id_policy in the same documents.

Conclusions: Alice and Bob comply with :ny_state_residency_policy. Alice and David comply with :ny_state_id_policy. (Alice and Bob stay in Troy, which is in NY state. Alice and David have NY state ids "307 578 001" and "307 578 002")

Tutorial Policy 19:

```
 @forAll :PERSON, :CITY, :NY_STATE_ID.
 :ny_state_residency_policy a air:Policy;
        air:rule :state-residency-rule.
 :state-residency-rule a air:Belief-rule;
     rdfs:label "state residency rule";
     air:pattern {
          :PERSON tamip:Lives_in_city :CITY.
          :CITY tamip:Has_state :NY.
     };
     air:assert {:PERSON air:compliant-with :ny_state_residency_policy.}.

 :ny_state_id_policy a air:Policy;
        air:rule :state-id-check.
 :state-id-check a air:Belief-rule;
     rdfs:label "state id check rule";
     air:pattern { :PERSON tamip:Has_ny_state_id :NY_STATE_ID. };
     air:assert {:PERSON air:compliant-with :ny_state_id_policy.}.
```

3.1.5.2. Multiple policies in more than one policy documents

The two policies in Tutorial Policy 19 in 3.1.5.1 are split over 2 files.
Conclusions: Alice and Bob comply with :ny_state_residency_policy. Alice and David comply with :ny_state_id_policy.

Tutorial Policy 9:

```
@forAll :PERSON, :CITY.
:ny_state_residency_policy a air:Policy;
       air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
    rdfs:label "state residency rule";
    air:pattern {
         :PERSON tamip:Lives_in_city :CITY.
         :CITY tamip:Has_state :NY.
    };
    air:assert {:PERSON air:compliant-with :ny_state_residency_policy.}.
```

Tutorial Policy 10:

```
@forAll :PERSON, :NY_STATE_ID.
:ny_state_id_policy a air:Policy;
       air:rule :state-id-check.
:state-id-check a air:Belief-rule;
    rdfs:label "state id check rule";
    air:pattern { :PERSON tamip:Has_ny_state_id :NY_STATE_ID. };
    air:assert {:PERSON air:compliant-with :ny_state_id_policy.}.
```

3.1.5.3. Variable Scoping over multiple policy documents

The Tutorial Policy 3 in 3.1.2.1.1 is split over 2 files. The second file contains only a rule- state-id-check rule, which is a nested in rule state-residency-rule. Here we see that variable bindings are retained when rules are cross-referenced across files.
Conclusions: Alice is compliant with ':ny_state_residency_and_id_policy'.

Tutorial Policy 17:
```
@forAll :PERSON, :CITY.
:ny_state_residency_and_id_policy a air:Policy;
    air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
    rdfs:label "state residency rule";
    air:pattern {
        :PERSON tamip:Lives_in_city :CITY.
        :CITY tamip:Has_state :NY.
    };
    air:rule :state-id-check.
```

Tutorial Policy 18:
```
@forAll :PERSON, :NY_STATE_ID.
:state-id-check a air:Belief-rule;
   rdfs:label "state id check rule";
   air:pattern { :PERSON tamip:Has_ny_state_id :NY_STATE_ID. };
   air:assert {:PERSON air:compliant-with :ny_state_residency_and_id_policy.}.
```

3.2 Justification

Every action is associated with justification(s) at the run time. By default the conjunction of matched graphs (antecedent) and the rule-id are given as justification for the actions. These justifications (i.e. antecedent & rule-id) can be explicitly modified or suppressed. A natural language description of the rule can also be provided.

3.2.1. Natural language explanation of the rule (air:description)

The policy is same as that in Tutorial Policy 1 in 3.1.1. Here, we also provide a natural language description for the :state-residency-rule, using air:description. Description is given as a sequence of variables and strings.

Conclusions: Bob and Alice comply with :ny_state_residency, and following descriptions: <:Bob> "lives in the NY state city -" <:Troy>, <:Alice> "lives in the NY state city -" <:Troy>.

```
Tutorial Policy 11:
@forAll :PERSON, :CITY.
:ny_state_residency_policy a air:Policy;
        rdfs:label "NY State residency policy";
        air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
     rdfs:label "state residency rule";
     air:pattern {
          :PERSON tamip:Lives_in_city :CITY.
          :CITY tamip:Has_state :NY.
     };
     air:description(:PERSON "lives in the NY state city -" :CITY);
     air:assert {:PERSON air:compliant-with :ny_state_residency_policy.}.
```

3.2.2. Hiding Justification (air:Hidden-rule)

The policy is same as Tutorial Policy 3 in 3.1.2.1.1. But we don't want the state id to show up in the justification. To hide it we declare :state-id-check to be a air:Hidden-rule.

Conclusions: Alice is compliant with :ny_state_residency_and_id_policy.

```
Tutorial Policy 12:
@forAll :PERSON, :CITY, :NY_STATE_ID.
:ny_state_residency_and_id_policy a air:Policy;
       air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
    rdfs:label "state residency rule";
    air:pattern {
         :PERSON tamip:Lives_in_city :CITY.
         :CITY tamip:Has_state :NY.
    };
    air:rule :state-id-check.
:state-id-check a air:Hidden-rule;
    rdfs:label "state id check rule";
    air:pattern { :PERSON tamip:Has_ny_state_id :NY_STATE_ID. };
    air:assert {:PERSON air:compliant-with :ny_state_residency_and_id_policy.}.
```

Below, we can see the difference between the two justifications. The justification for state-id-check rule does not appear in the latter case.

```
Justification for Tutorial Policy 3:
:justification    [
                    :antecedent-expr [
                           a :And-justification;
                           :sub-expr    [
                                  air:instanceOf <:state-id-check>;
                                  :justification    [
                          :antecedent-expr   [
                                 a :And-justification;
                                 :sub-expr <:state-residency-rule>,
                                             {<:Alice> <:Lives_in_city> <:Troy> .
                                               <:Troy> <:Has_state> <:NY> .} ];
                          :rule-name <:state-residency-rule> ] ],
                        {<:Alice> <:Has_ny_state_id> <:307_578_001> .
           } ];
        :rule-name <:state-id-check> ] .
```

Justification for Tutorial Policy 12:

```
:justification   [
                     :antecedent-expr   [
                            a :And-justification;
                            :sub-expr <:state-residency-rule>,
                            {<:Alice>  <:Lives_in_city>  <:Troy>.  <:Troy>  <:Has_state>
<:NY>.} ];
                     :rule-name <:state-residency-rule> ].
```

3.2.3. Explicit Justification (air:assertion, air:statement, air:Justification, air:antecedent, air:rule-id)

This policy is the same as Tutorial Policy 3 in 3.1.2.1.1 and Tutorial Policy 12 in 3.2.2. Again, we do not want the state id to show up in the justification. But instead of suppressing the justification we explicitly define the justification for the :state-id-check rule. The default rule-id :state-id-check is replaced by a new rule-id :state-residency-id-rule. The default antecedent is replaced by the matched-graph of :state-residency-rule, which is referenced through the variable :G1. Note that :G1 is bound whenever the :state-id-check rule is active. air:statement serves similar purpose as air:assert, i.e. the triple :PERSON air:compliant-with :ny_state_residency_and_id_policy, where :PERSON1 is replaced by its binding value, is asserted.

Conclusions: Alice is compliant with :ny_state_residency_and_id_policy. However, the justification produced is different from the justifications shown in the previous section.

Tutorial Policy 13:

```
@forAll :PERSON, :CITY, :NY_STATE_ID, :G1 .
:ny_state_residency_and_id_policy a air:Policy;
        air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
     rdfs:label "state residency rule";
     air:pattern {
          :PERSON tamip:Lives_in_city :CITY.
          :CITY tamip:Has_state :NY.
     };
     air:matched-graph :G1;
     air:rule :state-id-check.
:state-id-check a air:Belief-rule;
     rdfs:label "state id check rule";
     air:pattern { :PERSON tamip:Has_ny_state_id :NY_STATE_ID. };
     air:description ( :PERSON "is a new york state resident" );
     air:assertion [
          air:statement    {    :PERSON    air:compliant-
with :ny_state_residency_and_id_policy.};
          air:justification [
                 air:rule-id :state-residency-id-rule;
                 air:antecedent :G1
          ]
     ].
```

Justification for Tutorial Policy 13:

```
:justification   [
                 :antecedent-expr   [
                      a :And-justification;
                      :sub-expr <:state-residency-rule>,
                      {<:Alice>  <:Lives_in_city>  <:Troy>.  <:Troy>  <:Has_state>
<:NY>.} ];
                 :rule-name <:state-residency-rule> ].
```

3.2.4. Explanations for failed policy compliance (air:alt, air:non-compliant-with)

In section 3.1.3 we had seen how unmatched cases can be explicitly handled using air:alt, like in Tutorial Policy 16. This explicit handling also provides explanations for failed (non-compliant) policy decisions.

Conclusions: Alice and Bob comply with the :ny_state_residency_policy. George is not compliant with the :ny_state_residency_policy.

Justification for the conclusion <:George> air:non-compliant-with <:ny_state_residency_policy>:

```
:justification   [
     :antecedent-expr   [
          a :And-justification;
          :sub-expr   [
               air:instanceOf <#_g0>;
               :justification   [
                     :antecedent-expr   [
                                    a :And-justification;
                                    :sub-expr <:state-residency-rule>, {
                                                     <:George>                  <tamip:Lives_in_city>
<:Boston> .} ];
                             :rule-name <:state-residency-rule> ] ], [
                      air:closed-world-assumption    (
                                <:ny_state_residency_policy>                          air:base-rules
<:AIR_Demo_Data>                    <http://dig.csail.mit.edu/TAMI/2007/amord/base-
assumptions.ttl> );
                      :justification :premise ], {} ];
       :rule-name <#_g0> ].
```

4. Policy Engine Features

AIR policy reasoner is developed by Decentralized Information Group at MIT. It is an instance of Truth Maintenance System[6] and replied on an reasoner CWM[4].

4.1. OWL Inferencing Support

The policy engine supports reasoning for following owl (& rdfs) constructs:

- rdfs:subClassOf
- rdfs:subPropertyOf
- rdfs:domain
- rdfs:range
- owl:sameAs
- owl:TransitiveProperty
- owl:SymmetricProperty

The inference rules are encoded in AIR (base-rules).

The following policy is same as Tutorial Policy 4 in 3.1.2.1.2. However, the pattern of :neighbor-state-rule is changed- the triple :NY tamip:Neighbor_state :STATE is reversed. The data includes the triple :NY tamip:Neighbor_state :MA, only. But it also includes the triple tamip:Neighbor_state rdf:type owl:SymmetricProperty.
Conclusions: George is compliant with :ny_neighbor_state_residency_policy.
(George lives in Boston. Boston is in MA, not in NY. NY is neighbor state of MA, but since neighbor state is a symmetric property, MA is neighbor state of NY also holds.)

[4] CWM, Closed World Machine, http://www.w3.org/2000/10/swap/doc/cwm.html

Tutorial Policy 14:

```
@forAll :PERSON, :CITY, :STATE.
:ny_neighbor_state_residency_policy a air:Policy;
    air:rule :non-ny-residency-rule.
:non-ny-residency-rule a air:Belief-rule;
    rdfs:label "Non NY residency rule";
    air:pattern {:PERSON tamip:Lives_in_city :CITY.};
    air:rule [
        air:pattern {:CITY tamip:Has_state :NY.};
        air:alt [air:rule :neighbor-state-rule]
    ].
:neighbor-state-rule a air:Belief-rule;
    rdfs:label "neighbor state rule";
    air:pattern { :CITY tamip:Has_state :STATE.
                  :STATE tamip:Neighbor_state :NY.};
    air:assert { :PERSON air:compliant-with :ny_neighbor_state_residency_policy. }.
```

4.2. Multiple Logs

Consider a new data file AIR Demo Data1 (RDF)- contains single triple :Bill tamip:lives_in_city :Troy. The 2 data files, AIR Demo Data and AIR Demo Data1, are checked against Tutorial Policy 1.

Conclusions: Alice, Bob and Bill comply with :ny_state_residency policy. (Alice & Bob live in Troy, which is in NY. Bill also lives in Troy. Troy is in NY is stated in other log.)

4.3. Parameters can be passed to the engine

For example, a list of concluded predicates to be displayed in the reasoner's output can be passed to the engine. This is discussed in detail in 4.4.1.

4.4. Reasoner's output

Policy engine employs a forward chaining reasoner. On finding the deductive closure (of conclusions) the reasoner filters some conclusions to be displayed in the results, along with their justification. Other characteristics of the reasoners output:

- The result shows only compliance or non-compliance conclusions and contains justifications for each of these conclusions.
- In the current version, only 1 justification for each of the (non)compliance assertions are included.
- The results recursively include justifications for triples in the antecedent of the justification of conclusions that have been included so far, unless the triples are part of the assumptions (i.e. the triples are part of the data at the beginning)

The following policy is similar to the Tutorial Policy 1. We make an additional assertion that the person lives in state NY. We know that whenever a person is concluded to be compliant-with the :ny_state_residency_policy, it is also asserted that s/he lives in NY state.

Conclusions: Bob and Alice are compliant with :ny_state_residency policy. However, the result does not include the triples- <:Bob> tamip:Lives_in_state <:NY> and <:Alice> tamip:Lives_in_state <:NY>- that we know have been asserted as well.

Tutorial Policy 15:

```
@forAll :PERSON, :CITY.
:ny_state_residency_policy a air:Policy;
        air:rule :state-residency-rule.
:state-residency-rule a air:Belief-rule;
    rdfs:label "state residency rule";
    air:pattern {
        :PERSON tamip:Lives_in_city :CITY.
        :CITY tamip:Has_state :NY.
    };
    air:assert {:PERSON air:compliant-with :ny_state_residency_policy.
                    :PERSON tamip:Lives_in_state :NY.    }.
```

4.4.1. Display all conclusions for a predicate

By default the policy engine chooses to filter all the conclusions with the predicates air:compliant-with and air:non-compliant-with. We can pass a list of URIs of predicates to the argument named filterProperties, directing the policy engine to filter all the conclusions with these predicates (in addition to air:compliant-with and air:non-compliant-with predicates).

We will run dataset by Tutorial Policy 15, and pass tamip:Lives_in_state as argument to filterProperties.

Conclusions: <:Bob> air:compliant-with :ny_state_residency, <:Alice> air:compliant-with :ny_state_residency, <:Bob> tamip:Lives_in_state <:NY> and <:Alice> tamip:Lives_in_state <:NY>

5. Discussion

In this tutorial we have covered many salient features and functionalities of AIR language and an AIR policy engine respectively. Even though writing rules in AIR is quite simple but it is error prone. It is easy to make mistakes such as encoding rules that are unsafe or set of rules that make contradictory compliance decisions under some circumstances. While it is easy to detect unsafe rules before actual policy reasoning, the same cannot be said about the latter. One of the future tasks is to devise a mechanism to support off-line detection of rules that could make contradictory decisions. Further, a very visible gap in current AIR language is the lack of support for datatypes. Since CWM supports built-in functions that may be used for comparisons and other computations on numbers and strings, it is worth exploring datatype support in context of AIR.

Acknowledgments. We thank for valuable help from our colleagues at Tetherless World Constellation at RPI and Decentralized Information Group at MIT.

References

1. Lalana Kagal, Chris Hanson, and Daniel Weitzner. Using Dependency Tracking to Provide Explanations for Policy Management. IEEE Policy, June 2008
2. David Beckett. Turtle - Terse RDF Triple Language, http://www.dajobe.org/2004/01/turtle/, last modified on 20 November 2007
3. Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3Logic: A Logical Framework for the World Wide Web. Journal of Theory and Practice of Logic Programming (TPLP), Special Issue on Logic Programming and the Web, 2007
4. AIR Policy Language, http://dig.csail.mit.edu/TAMI/2008/12/AIR/, last modified on 10 Dec 2008
5. Dan Brickley, R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema (W3C Recommendation 10 February 2004) http://www.w3.org/TR/rdf-schema/
6. J. Doyle. A truth maintenance system. Artificial Intelligence, 12(3):231–272, November 1979.