# Accessible Reasoning

## What Comes After Pattern-Matching and Tracing in AIR?

**Ian Jacobi**
MIT

**Ankesh Khandelwal**
RPI



# Where is AIR moving?

- Increasing comprehensibility
- Explicitly defining justification semantics
- Extending pattern-matching with built-ins
- Moving beyond the Closed World

# Revising the Syntax

- Revised old `air:pattern`, `air:rule`, `air:alt` syntax for ease of use.
  - `air:pattern` → `air:if`
  - `air:rule` → `air:then [ air:rule ]`
  - `air:assert` → `air:then [ air:assert ]`
  - `air:alt` → `air:else`

# Example in old syntax

```
:CheckInfringement a air:Belief-rule ;
    air:pattern {
        :Violation a
copyright:PotentialCopyrightInfringement };
    air:rule :CompareDates ;
    air:alt [ air:assert {
        :Violation air:non-compliant-with
            :CopyrightPolicy . } ] .
```

# Example in new syntax

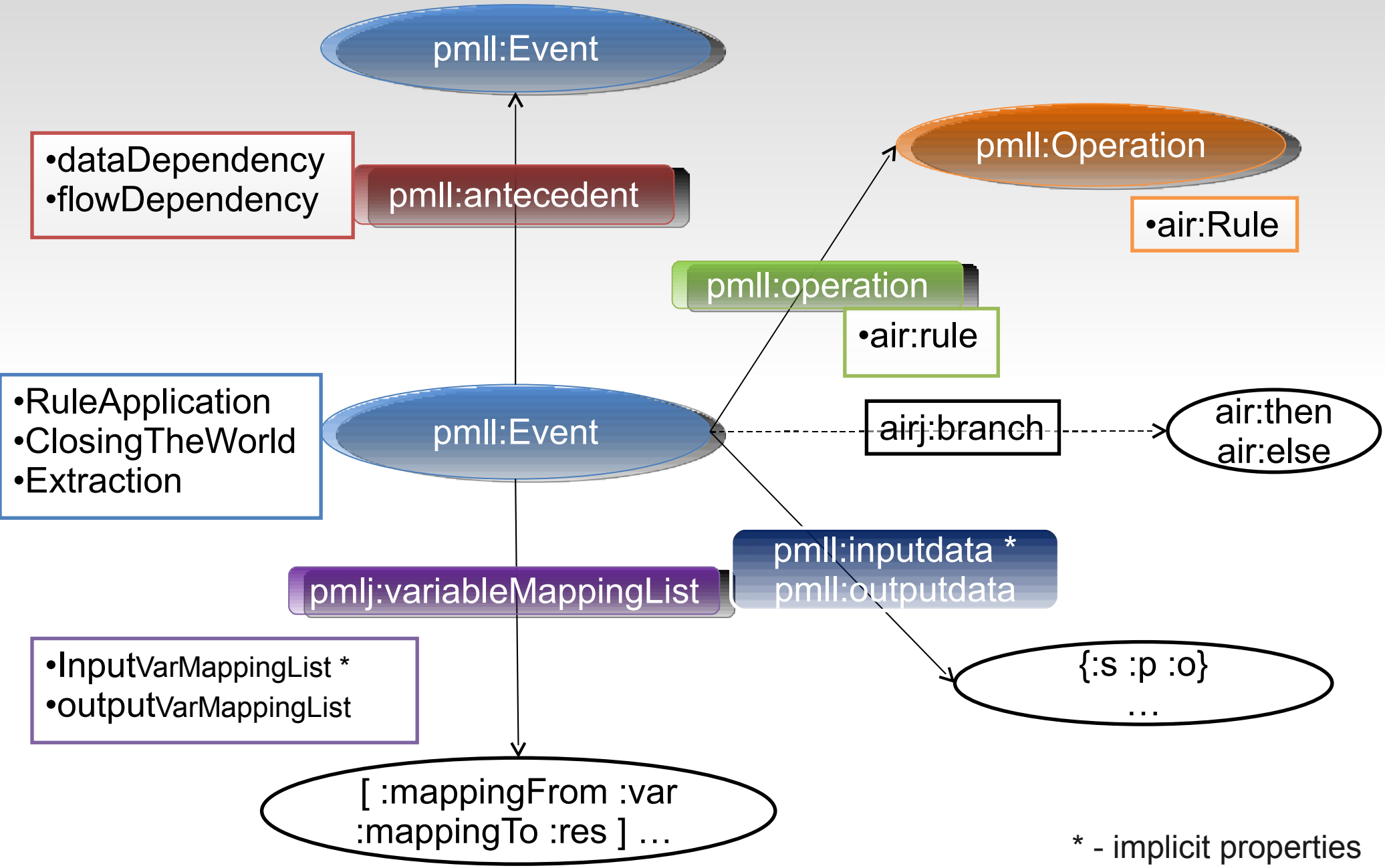
```
:CheckInfringement a air:Belief-rule ;
    air:if {
        :Violation a
copyright:PotentialCopyrightInfringement };
    air:then [ air:rule :CompareDates ] ;
    air:else [ air:assert {
        :Violation air:non-compliant-with
            :CopyrightPolicy . } ] .
```

# Current Justifications are muddy...

```
:MyVio air:compliant-with :CopyrightPolicy .
:CheckInfringement tms:justification tms:premise.

{:MyVio air:compliant-with :CopyrightPolicy.}
    tms:justification [
        tms:antecedent-expr [
            a tms:And-justification;
            tms:sub-expr :CheckInfringement ,
                {:MyVio a
copyright:PotentialCopyrightInfringement . },
                [ ... ] ] ;
        tms:rule-name :CheckInfringement ] .
```

# Justifications Using PMLLite




* - implicit properties

# Justification Example - Policy

```
@forAll :Violation, :Work, :Creator, :ViolationDate, :DeathDate .

:CopyrightCriminalPolicy a air:Policy ;
            air:rule :FindInfringement .

:FindInfringement a air:Belief-rule ;
    air:if {
         :Violation a copyright:PotentialCopyrightInfringement ;
                    copyright:infringesCopyrightOn :Work . } ;
    air:then [ air:rule :FindValue ] .

:FindValue a air:Belief-rule ;
    air:if {
        :Work gr:hasCurrencyValue :Value ;
              gr:hasCurrency "USD" . } ;
    air:then [ air:rule :CheckValue ] .

:CheckValue a air:Belief-rule ;
    air:if {  :Value math:lessThan "1000" . } ;
   air:then [ air:assert { :Violation air:non-compliant-with
                      :CopyrightCriminalPolicy . } ];
              air:description ( :Violation " is not a criminal copyright infringement as
   it is under $1,000 in value" ) ] ;
```

# Justification Example - Log

```
:MinorInfringement a
   copyright:PotentialCopyrightInfringement ;
     copyright:infringesCopyrightOn :SpaceOdyssey ;
     copyright:infringementDate "2008-10-01" .

:SpaceOdyssey a movie ;
     gr:hasCurrencyValue "30" ;
     gr:hasCurrency "USD" ;
     dc:creator :StanleyKubrick .

"30" math:lessThan "1000" . *
```

URI : <http://dig.csail.mit.edu/TAMI/inprogress/example-log.n3>

* Consider this fact to be a premise.

# Justification Example – Justification (1)

```
pr:Event4 a airj:RuleApplication;
    air:rule :CheckValue;
    pmll:outputdata { :MinorInfringement air:non-compliant-with
    :CopyrightCriminalPolicy . };
    air:description ":MinorInfringement is not a criminal copyright infringement as it is
    under $1,000 in value";
    airj:branch air:then;
    airj:dataDependency pr:GetDataLog ;
    airj:flowDependency pr:Event3 .

pr:Event3 a airj:RuleApplication;
    air:rule :FindValue;
    airj:outputVariableMappingList (pr:Mapping3, pr:Mapping2, pr:Mapping1);
    airj:branch air:then ;
    airj:dataDependency pr:GetDataLog ;
    airj:flowDependency pr:Event2 .

pr:Mapping3 a pmlj:Mapping;
     pmlj:mappingFrom :Value;
     pmlj:mappingTo   "30".
```

# Justification Example – Justification (2)

```
pr:inputLog is log:semantics of <http://dig.csail.mit.edu/TAMI/inprogress/example-log.n3>.

pr:GetDataLog a airj:Extraction;
    pmll:outputdata pr:inputLog.

pr:Event2 a airj:RuleApplication;
    air:rule :FindInfringement;
    …
    airj:flowDependency pr:Event1 .

pr:Event1 a airj:RuleApplication;
    air:rule :CopyrightCriminalPolicy;
    …

pr:Mapping2 a pmlj:Mapping;
    pmlj:mappingFrom :Work;
    pmlj:mappingTo   :SpaceOdyssey.

pr:Mapping1 a pmlj:Mapping;
    pmlj:mappingFrom :Violation;
    pmlj:mappingTo   :MinorInfringement.
```

# Ellipsed Justification Example - Policy

```
@forAll :Violation, :Work, :Creator, :ViolationDate, :DeathDate .

:CopyrightCriminalPolicy a air:Policy ;
            air:rule :FindInfringement .

:FindInfringement a air:Belief-rule ;
    air:if {
          :Violation a copyright:PotentialCopyrightInfringement ;
                     copyright:infringesCopyrightOn :Work . } ;
    air:then [ air:rule :FindValue ] .

:FindValue a air:Ellipsed-rule ;
    air:if {
        :Work gr:hasCurrencyValue :Value ;
              gr:hasCurrency "USD" . } ;
    air:then [ air:rule :CheckValue ] .

:CheckValue a air:Belief-rule ;
    air:if {  :Value math:lessThan "1000" . } ;
    air:then [ air:assert { :Violation air:non-compliant-with
                        :CopyrightCriminalPolicy . } ] ;
               air:description ( :Violation " is not a criminal copyright infringement as it is
    under $1,000 in value" ) ] ;
```

# Ellipsed Justification Example – Justification

```
pr:Event4 a airj:RuleApplication;
    air:rule :CheckValue;
    pmll:outputdata { :MinorInfringement air:non-compliant-with
    :CopyrightCriminalPolicy . };
    air:description ":MinorInfringement is not a criminal copyright infringement as it is
    under $1,000 in value";
    airj:branch air:then;
    airj:dataDependency pr:GetDataLog ;
    airj:flowDependency pr:Event3 .

@forSome pr:Event3 .
pr:Event3 a airj:RuleApplication;
    air:rule :FindValue;
    airj:outputVariableMappingList (pr:Mapping3, pr:Mapping2, pr:Mapping1);
    airj:branch air:then ;
    airj:dataDependency pr:GetDataLog ;
    airj:flowDependency pr:Event2 .

pr:Mapping3 a pmlj:Mapping;
     pmlj:mappingFrom :Value;
     pmlj:mappingTo   "30".
```

# Hidden Justification Example - Policy

```
@forAll :Violation, :Work, :Creator, :ViolationDate, :DeathDate .

:CopyrightCriminalPolicy a air:Policy ;
            air:rule :FindInfringement .

:FindInfringement a air:Belief-rule ;
    air:if {
         :Violation a copyright:PotentialCopyrightInfringement ;
                    copyright:infringesCopyrightOn :Work . } ;
    air:then [ air:rule :FindValue ] .

:FindValue a air:Hidden-rule ;
    air:if {
        :Work gr:hasCurrencyValue :Value ;
              gr:hasCurrency "USD" . } ;
    air:then [ air:rule :CheckValue ] .

:CheckValue a air:Belief-rule ;
    air:if {  :Value math:lessThan "1000" . } ;
   air:then [ air:assert { :Violation air:non-compliant-with
                      :CopyrightCriminalPolicy . } ] ;
              air:description ( :Violation " is not a criminal copyright infringement as
   it is under $1,000 in value" ) ] ;
```

# Hidden Justification Example – Justification

```
@forSome pr:Event4 .
pr:Event4 a airj:RuleApplication;
    air:rule :CheckValue;
    pmll:outputdata { :MinorInfringement air:non-compliant-with
    :CopyrightCriminalPolicy . };
    air:description ":MinorInfringement is not a criminal copyright infringement as it is
    under $1,000 in value";
    airj:branch air:then;
    airj:dataDependency pr:GetDataLog ;
    airj:flowDependency pr:Event3 .

pr:Event3 a airj:RuleApplication;
    air:rule :FindValue;
    airj:outputVariableMappingList (pr:Mapping3, pr:Mapping2, pr:Mapping1);
    airj:branch air:then ;
    airj:dataDependency pr:GetDataLog ;
    airj:flowDependency pr:Event2 .

pr:Mapping3 a pmlj:Mapping;
     pmlj:mappingFrom :Value;
     pmlj:mappingTo   "30".
```

# Extend Pattern-Matching with Built-in Functions

- Pattern-matching alone can't solve some problems
  - “Has the author been dead more than 70 years?”
  - “Is the infringement over $1,000 in value?”
  - “Was the item actually under copyright?” (without re-encoding copyright rules)
- Extend pattern-matching by introducing built-in functions (`math:`, `time:`, `log:`...)
- Can even do “meta-reasoning” (`air:justifies`)

# Built-in Functions

crypto:md5
crypto:sha
crypto:keyLength
crypto:sign
crypto:verify
crypto:verifyBoolean
crypto:publicKey
**list:in**
**list:member**
**list:last**
**list:append**
**list:members**
**math:sum**
**math:difference**
**math:product**
**math:quotient**
**math:integerQuotient**
**math:remainder**
**math:exponentiation**

math:sumOf
math:differenceOf
math:factors
math:bit
math:quotientOf
math:remainderOf
math:exponentiationOf
**math:negation**
**math:absoluteValue**
**math:rounded**
**math:greaterThan**
**math:notGreaterThan**
**math:lessThan**
**math:notLessThan**
**math:equalTo**
**math:notEqualTo**
math:memberCount
**fn:abs**
**fn:ceiling**

**fn:floor**
**fn:round**
**fn:round-half-to-even**
**set:in**
**set:member**
**set:union**
**set:intersection**
**set:symmetricDifference**
**set:difference**
**set:oneOf**
**string:greaterThan**
**string:notGreaterThan**
**string:lessThan**
**string:notLessThan**
**string:startsWith**
**string:endsWith**
**string:concat**
string:concatenation
string:scrape

**BOLD** are probable candidates for standardization.

# Built-in Functions

**string:search**
**string:split**
string:stringToList
**string:matches**
**string:notMatches**
**string:contains**
**string:containsIgnoring Case**
**string:containsRoughly**
**string:doesNotContain**
**string:equalIgnoringCase**
**string:notEqualIgnoring Case**
string:xmlEscape Attribute
string:xmlEscapeData
**string:encodeForURI**
**string:encodeForFragID**
fn:resolve-uri

fn:tokenize
fn:normalize-space
fn:codepoints-to-string
fn:string-to-codepoints
**fn:compare**
fn:codepoint-equal
**fn:concat**
**fn:string-join**
**fn:substring**
**fn:string-length**
fn:normalize-unicode
**fn:upper-case**
**fn:lower-case**
fn:translate
**fn:encode-for-uri**
fn:iri-to-uri
**fn:escape-html-uri**
**fn:contains**
**fn:starts-with**

**fn:ends-with**
**fn:substring-before**
**fn:substring-after**
**fn:matches**
**fn:replace**
**time:inSeconds**
**time:year**
**time:month**
**time:day**
**time:date**
**time:equalTo**
**time:hour**
**time:minute**
**time:second**
**time:dayOfWeek**
**time:timeZone**
**time:gmTime**
**time:localTime**
**time:format**

**BOLD** are probable candidates for standardization.

# Built-in Functions

**time:formatSeconds**
**time:parseToSeconds**
**math:cos**
**math:cosh**
**math:degrees**
**math:sin**
**math:sinh**
**math:tan**
**math:tanh**
xml:nodeType
xml:parentNode
xml:attributes
xml:previousSibling
xml:nextSibling
xml:childNodes
xml:firstChild
xml:lastChild
xml:localName
xml:prefix

xml:namespaceURI
xml:nodeName
xml:nodeValue
xml:hasAttributes
xml:hasChildNodes
xml:xpath
fn:string
fn:doc
sparql:equals
sparql:lessThan
sparql:greaterThan
sparql:notGreaterThan
sparql:notLessThan
sparql:notEquals
sparql:typeErrorIsTrue
sparql:typeErrorReturner
sparql:truthValue
sparql:lamePred
sparql:query

sparql:semantics
sparql:dtLit
sparql:langLit
log:racine
log:dtlit
log:rawType
log:rawUri
log:filter
log:vars
log:universalVariable Name
log:existentialVariable Name
log:enforceUnique Binding
log:conjunction
log:uri
log:equalTo
log:notEqualTo

**BOLD** are probable candidates for standardization.

# Built-in Functions

**owl:sameAs**
**log:includes**
**log:supports**
**log:notIncludes**
log:notIncludesWith
    Builtins
**log:semantics**
log:semanticsOrError
log:semanticsWith
    ImportsClosure
log:content
log:parsedAsN3
log:n3ExprFor
log:n3String
log:definitiveService
**air:justifies**
maths: functions
times: functions
os: functions

**BOLD** are probable candidates for standardization.

# Example built-in functions

- `(1 2) math:sum ?x .`
  - `?x = 3 .`
- `"2009-10-30" time:year ?y .`
  - `?x = "2009" .`
- `:a list:in (:a :b :c) .`
  - (Will match)
- `:z list:in (:a :b :c) .`
  - (Won't match)

# Example using built-ins

```
:CompareDates a air:Belief-rule ;
    air:if {
        :DeathDate time:year :DeathYear .
        :ViolationDate time:year :ViolYear .
        (:DeathYear "70") math:sum :ExpYear .
        :ViolYear math:lessThan :ExpYear .};
    air:then [ air:assert [
        air:statement {
            :Work air:compliant-with
                :CopyrightPolicy . } ] ] .
    air:else [ air:assert [
        air:statement {
            :Work air:non-compliant-with
                :CopyrightPolicy . } ] ] .
```

# Example using air:justifies

```
:CheckApplicableCopyright a air:Belief-rule ;
    air:if {
        @forSome :log, :rule .
        <./data.n3> log:semantics :log .
        <./copyright.n3> log:semantics :rule.
        ( ( :log ) ( :rule ) ) air:justifies
          { <./data.n3#Work>
            air:compliant-with
            :CopyrightPolicy . } } .
    air:then [ air:assert [ air:statement {
        :Violation air:compliant-with
        :CriminalInfringementPolicy . } ] ] .
```

# Future Work: Moving beyond the closed-world: Assumption-based reasoning

- Reasoner currently assumes a black/white world
  - Either you can prove something true, or, it's not true.
- Reasoner can't handle contradicting sources
- Solution: Assumption-based reasoning
  - Use TMS more fully
    - Assume open/closed world as needed
    - Maintain contradicting sources and indicate them
  - Introduce `air:null` to “hesitate” in decision-making
    - Open-world assumption instead of closed-world of `air:else`

# Questions?