このドキュメントは
The Semantic Toolbox: Building Semantics on top of XML-RDF
http://www.w3.org/DesignIssues/Toolbox.html
の和訳です。
このドキュメントには和訳上の誤りがありえます。
内容の保証はいたしかねますので、必ず W3C Web サイトの正式版ドキュメントを参照して下さい。

状態: 個人的メモ, 多くの部分が未完成。一貫性を保持する要求を放棄した。最初に作られた日: 1999/06/18

このドキュメントは RDF の 3 つ組みの抽象的な構文の上に論理式を構築しようと試みている。他のより最近の研究には、Notation3 がある。それは、論理のために XML を使う研究である。

このドキュメントには、以下のものが含まれる

- 仮定する構文
- 意味論の文脈(Semantic Context)
- 他のドキュメントの宣言(include, assert, truth)
- 論理式：not
- 信頼文の例
- 限量子(Quantification)(for all, exists)
- ・・・

The Semantic Toolbox: XML-RDF の上に意味を与える

XML の構文や RDF モデルは Web の基本的な意味を与える。しかし、<some conective tissue to work towark the semantic web>について考える必要があるように思える。
基本的に、Web 上の「データ」と考えられているものはどれも、論理文(logical statements)の形になっている。情報システムに対するヒューマンインタフェースを統一的に扱う（記述）言語として HTML があるように、データシステムに対するマシンインタフェースとしてデータに対する統一的な論理言語を定める必要がある。
このドキュメントでは
この文書は、コミュニティで提案された多数の要求条件(requirements)を満足する妥当な XML/RDF モデルの存在証拠の試みである。
これには以下が含まれる；

- efinitive なリストからはじめる
- Making a reference contingent on a digital signature of the destination;
- Writing inference rules for linking old and new schemata;
- Expressing the equivalence of terms in different database schemas;
- ...

要求(The need)はもちろん、できるだけ自然に、RDF の中に論理を構築することである。
機能を追加するごとに構文が劇的に複雑になるようでは失敗である。逆に、もし、高階の(higher-oder) Statement が単純なメタデータの宣言と同様に XML で自然に記述できれば成功といえる。各段階で、英語（自然言語）でしか表記できないような意味を持つ特殊な XML 構文を導入するようなら失敗だが、RDF のプロパティの導入だけで言語が構築できれば、特にその意味論が RDF やそれ以前に導入されたプロパティを用いて記述できるようなら、成功である。

(このドキュメントでは、名前空間のプレフィックス付きの XML 要素は、読者がそれとわかるようなものを仮定しており、プレフィックスなしのものは、ここで導入する新しい機能を用いるために導入された要素である。).

仮定する構文

ここでの目的のために、データのための構文は別途規定した XML での形式と仮定する。

Semantic Context**（意味の文脈）**

宣言文(Assertions)は全て等しい訳ではない。異なるドキュメントで異なる人々が異なる程度に保証したものでしかない。それらは、参照されたり、あるいは、明示的に否定されたりすることもある。従って、その宣言文の文脈(context)は、その使用にとって不可欠なものとなる。

文脈は、ネストした XML 要素(element)の中で、それを変更する要素が出てこない限り継承される。

宣言文が検証(verified)される時、その正確さの証拠は蓄積され、信頼評価の主観的基準として提出(submit)される。究極的な信頼基準が主観的である以上、データが Web 上に置かれる時、それの意味を与える論理は、正に曖昧性なく、Well Defined でなければならない。

On reification

現在の RDF モデルは、宣言文の（非順序(unordered)）集合である。

すべての新しい特色は、新しい RDF プロパティとして導入されることができる。しかしながら、文の一部を述べるために、直接書かずに、RDF を間接的に使用するとき、我々はこれが実際に処理情報の実際的でない方法であることがわかるだろう。この（具体化(reification)とよぶ）プロセスは、RDF のモデルと構文(the RDF Model & Syntax)と題するドキュメントに記述されている。 （ひとつの）RDF 文は、モデルの中で４つの 3 つ組み(triple)として具体化(reify)される。３つは、宣言文の中の主語(subject)、目的語(object)そして述語(predicate)を宣言するのに必要である。もう一つは、その 3 つ組みが与えられたモデル（3 つ組みの集合）の一部であることを宣言するために用いられる；ここで一つ以上のモデルが存在し得る。従って、具体化(reification)は、必要とする記憶容量を 4 倍に跳ね上げる。

Refication が、連続的に饒舌な応答を返して、全く受け入れがたくなってしまうのが判るだろう。従って、新しい RDF プロパティの定義により、新しい言語機能を引き出すと同時に、実際的に有益になるよう、新しい言語が冗長にならないための構文も必要となる。

Reification は、明示的な文を明確に宣言されないが、記述したり言及したりすることのできる文の記述に変化させる。言語の中では、このことは典型的には引用で起こる。現在までのところ RDF の構文では、これを行うことができない(now way を no way の誤りとして訳)ので、ここから始めよう。

*Quotation**（引用）***

宣言する必要の無い宣言文を許すための特定の要素(element)はまだないので、QUOTE を仮定しよう。“ ラルフはオーラが本を書いたと言った ”という例題では、“ オーラが本を書いた ”は明らかに引用されている。我々が言及していることと、我々が言及しようとしている文とを区別する必要がある。

このことは、多くの情報源(source)からの情報を組み合わせる Web 上ではきわめて重要である。言語設計の基本的な部分とも言える。（たとえば、PICS ラベルシステムはこれを用いている。情報に関する情報であるメタデータでは、引用は明らかに本質的である。）

ひとつの方法は、

```
<quote id=foo about="theBook">
        <dc:author>Ora</dc:author>
</quote>
<rdf:description href="#foo">
   <dc:author>Ralph</dc:author>
   <http:from>swick@w3.org</http:from>
</rdf:description>
```

ここで、引用部は、オーラが本を書いたと言っている、そして続く記述で、ラルフがそれを言ったことを宣言している。これ自身をラルフが書いた訳ではなく、従って現在の著者は、引用の内容が正しいともそうでないとも主張していない事に注意しよう。

```
<quote id=foo about="theBook">
    <rdf:description href="#foo">
      <dc:author>Ralph</dc:author>
      <http:from>swick@w3.org</http:from>
    </rdf:description>
    <dc:author>Ora</dc:author>
</quote>
```

もし、“ 私の同封源要素(about my enclosing parent element) ” としての<head> 要素が広く認知されているならば、以下のように略記することもできる。

```
<quote about="theBook">
    <head>
      <dc:author>Ralph</dc:author>
      <http:from>swick@w3.org</http:from>
      <follows-from>http://www.org/catalog</follows-from>
    </head>
    <dc:author>Ora</dc:author>
</quote>
```

実際、含んでいるものを宣言しない点を除き、基本的に <quote> は <rdf:Description> と同じである。.[RDF ワーキンググループの検討結果の私の理解によれば].

他のドキュメントの宣言

(2000 年 10 月の daml:imports および Dan's GET/PUT model も参照のこと)

ドキュメント内の直接宣言された宣言文の集合の中から宣言文を除外することが重要であるように、ドキュメント内に存在しない宣言文を含めることも重要である。これは他のプロパティにより簡単に行える。結局、これは、A を信じるならの一環として B も信じることを示している一つの宣言なのである。

```
<foo:bar>
    <head>
```

```
      <include rdf:value="part1.rdf" />
      <include rdf:value="part1.rdf" />
      <include rdf:value="part1.rdf" />
    </head>
  </foo:bar>
```

このドキュメントは、Type はなんでもよいのだが、意味的には、part1.rdf, part2.rdf, and part3.rdf の情報を含むと考えられる。ここで HEAD を使っているのは、略記である。

（これは、文字的なインクルードではない- 他のドキュメントから意味を持ってくるだけであり、この場所でパースされる訳ではない。もし、対象とするドキュメントが SMIL の HTML 記述を含んでいる場合でも、人間が見るテキストや画像は起動されず、ここには表示されない。）

ここでは、何故あるいはどのようにこれらのドキュメントを信頼するかについての情報は含まれていない。この文は単にドキュメントの意味についての言及でしかない。言語の中で、意味とその信頼性を分離することは重要である。

（これらのために、人々に理解してもらう、この非常に基本的な論理的機能(logical function)の名前を決めるのも結構大変である。“ 保証する(Vouch) ” は “ 真実を宣言する ” の意味で良いし、“ 含意する(Imply) ” も 2 つのドキュメントで、一つのドキュメントを信じないなら、2 つ目のドキュメントも信じないという関係が成り立つなら良い名前だ。“ 宣言(Assert) ” や “ 真である(IsTrue) ” も使えそうである。）

保証された文書と現在の文書の間を二項関係として表現することは、複雑すぎる。現在の文書に、単項の関係 true(f) として表現する。それには、XML の略記よりは、RDF のプロパティで十分である；ちょっと汚いがより単純である。

```
  <foo:bar>
      <assert href="part1.rdf" />
      <assert href="part2.rdf" />
      <assert href="part3.rdf" />
  </foo:bar>
```

代わりに、そのドキュメントの真偽値の文を作ることもできる。

```
  <rdf:description about="part1.rdf">
    <truth>1</truth>
  </rdf:description>
```

これも、素直に読める。”1”の代わりに”0”だと言うなら、どうなるか？

```
  </quote>
  <rdf:description about="#foo">
    <truth>0</truth>
  </rdf:description>
```

論理式：否定（NOT）

人間が読むためのドキュメントの論理式については、もちろん MathML があるけれども、我々はセマンティック Web のための論理式の形式はまだ持っていない。論理式の対する実際的な必要性は IETF の “ connng ” グループの作業や、W3C のアクセス制御に関する internal work でも明らかになってきている。

当然、並置による “ AND ” はすでに使われている。前後に現れる 2 つの文は、その文脈で同じ外延として信頼される。これらをいっしょに解釈しないような論理システムは考え難い。だから、

```
{ S1, S2 } == S1 & S2
```
でありこれは“ & ”で定義される、並置は既に存在していてるので論理積は既に存在している。

もっとも簡単な論理式一つは NOT(x)だが、これは、XML では以下の XML 要素で自然にマップされる。

```
<bar id="foo" about="http://ww.w3.org/">
    <w3c:member>http://www.ibm.com/</w3c:member>
    <not>
        <w3c:member>http://www.soap.com/</w3c:member>
    </not>
</bar>
```

not は目的語(subject)に対しては影響を与えない(transparent)が、明らかに信頼に対してはそうではない。それは、含まれる宣言が偽であることを明示する宣言である。

具体化(Reification)**による否定**(Not)

RDF の三つ組みに直接基づく、構築中の言語を処理するマシンを提案する訳ではないが、basic RDF への新しい機能を具体化(ground)することは重要である。RDF はその basic level ではほとんどパワーがないので、RDF に何か記述して、具体化(reification)により何か新しいものを導入する必要がある。従って、"not(node, property, value)"と書くと、例えば“ RDF の中に、A が主語で、B プロパティの値が値 C であるような RDF プロパティがあるというのは偽である ”といったことを言う必要がある。そうすると RDF の中で、他のノードと真偽値を結びつける新しいプロパティとして否定を導入することができる。勿論、実際には、このように情報を扱うのは効率的ではない。

```
<quote id="foo" about="http://www.w3.org/">
    <w3c:member>http://www.soap.com/</w3c:member>
</quote>
<rdf:description about="#foo">
   <truth>0</truth>
</rdf:description>
```

<include>と意味的に重複している。

つまり、否定を含む式の表現は 2 とおりある。3 つ組みの集合だけを許す厳密な RDF の作法では、上記の reification を用いる。拡張モデルを用いる方法では、単純にそれぞれを符号化する。
否定が導入されなければ、RDF データベースの各宣言は、独立に扱う事ができ、事実(fact)の消去しても、真でない(untruth)という情報を作り出すことはない。
しかしながら、否定(not)があると、否定を含む and の中で項(term)の全集合を知る必要があり、式は何でも推論できるようになるので、真でないという情報を作り出すことができる。
否定は、非常に強力である。否定(not)と論理積(and)が与えられると、論理学者とゲート設計者が知るように、多くのことができる。否定要素の内容が論理積で繋がれ(Anded)ていれば、“ NAND ”関数を持つことになる。
[“Nand” は、所謂シェファーの棒(Sheffer’s stroke)であり、完全な命題論理系を単一で構築できる演算子であることが 1913 年に示され、

1970 年代には、7400 シリーズの基本ビルディングブロックとして用いられた。]
NAND を用いることにより論理和(OR)は以下のように構成される。

```
<not>
    <not>
        <w3c:member>http://www.ibm.com/</w3c:member>
    </not>
    <not>
        <w3c:member>http://www.soap.com/</w3c:member>
    </not>
</not>
```

これは、“ IBM は W3C のメンバーであるか、soap.com が W3C のメンバーであるかのどちらかである ” と同じである。これは少し格好が悪いが、同義語を用いて以下のように書くと少しは自然になる。

```
<alternatives>
    <or>
        <w3c:member>http://www.ibm.com/</w3c:member>
    </or>
    <or>
        <w3c:member>http://www.soap.com/</w3c:member>
    </or>
</alternatives>
```

含意(Implication)も否定を用いて構成できる。“ もし、soap.com がメンバーなら IBM もメンバーだ ” は、“ もしも、soap.com がメンバーであるって、IBM がメンバーでないこと無い ” と書ける、即ち、

```
<not>
    <w3c:member>http://www.ibm.com/</w3c:member>
    <not>
        <w3c:member>http://www.soap.com/</w3c:member>
    </not>
</not>
```

これは、同様に人間の読み手に合わせるなら、否定の同義語を用いて以下のように書ける。

```
<if>
    <w3c:member>http://www.ibm.com/</w3c:member>
    <then>
        <w3c:member>http://www.soap.com/</w3c:member>
    </then>
</if>
```

信頼文(trust statement)の例

上記で、他のドキュメントの意味を<include>する例をあげた。ある場合には、読み手(reader)を保護するために、単純な呼び出しに制約をかけたいと思う場合もあるだろう。このために、例えば、特定のチェックサムや電子署名を要求して条件付にすることができる。

Example of trust statemement

```
<foo:bar>
```

```
  <head>
    <if>
      <ds:hash rdf:about=part1.rdf">
        md5:1287129371237..12738127398712</ds:hash>
      <then>
          <include rdf:value="part1.rdf" />
      </then>
    </if>
    <if>
      <ds:signed-by rdf:about=part2.rdf">
        rsa:a/1024/123hg1238912whh3983yd2734dg
      </ds:signed-by>
      <then>
          <include rdf:value="part2.rdf" />
      </then>
    </if>
  </head>
</foo:bar>
```

ここで、ドキュメントは、part1 の内容は、それが適切なハッシュ値を持っているならば宣言(assert)され、part2 の内容は、それが特定の公開鍵で検証される電子署名を持っているならば宣言(assert)される。(ds は、hash と signed-by が存在すると仮定する名前空間であるが、ここではハッシュ値としては URI md5 スキームが存在することおよび、RSA 鍵も URI と見なしていることを指摘して、これ以上は議論しない。)
この機能(functionality)が存在する機能(features)を用いて達成されているのはなんとすばらしいことだろう。もう少しコンパクトにうまくやる方法はわからないが、２つの文は少し冗長かもしれない。

限量演算（Quantification）

上記の例は、特殊すぎるが、実際にはより汎用な記多くの規則がある。XML に限量記号、“ 全ての ”(”for all”)や“ 在る ”(”there exists some”)を如何に加えるのが良いだろうか？他のものと同様に、RDF の中に具体化の段階(reifiying)で（数式の構造を記述し、その構造に関する何らかの情報を宣言することにより）入れることもできる。つまり、形式的には退屈な具体化(reification)により、以下のように書くことができる。

```
  <quote id="foo" about="http://www.w3.org/">
     <w3c:member>http://www.soap.com/</w3c:member>
  </quote>
  <rdf:description about="#foo">
    <true-for-all>http://www.soap.com/</true-for-all>
  </rdf:description>
```

この例題では、（上記の否定の具体化(reification)に比べ）“ W3C は soap をメンバーに持っている ” は識別子 #foo が与えられていて、"http://www.soap.com/"　をどのような値に書き換えても真であると宣言している。これは、限量記号を使う直感的な方法ではなく、変数の名前も奇妙かもしれないが、"true-for-all"という一つ

のプロパティを追加しただけで限量演算が導かれている。Note

XML による論理のための限量記号の構文

これをどのように XML ベースのツールボックスに導入するかは自明ではない。XML の上にかぶせるか、XML を拡張するかの方法がある。ここでは、XML の上にかぶせる例を見てみる。Forall 節のための XML 要素を使うと同時に、変数のために XML の ID 空間を用いる。節内での変数に対する参照は、ID に対する参照で行われる。

```
<forall id="baz" var="x" rdf:about="#x">
  <if>
    <w3c:memberOf>http://www.w3.org/</w3c:memberOf>
    <then>
       <w3c:canAccess>http://www.w3.org/Member</w3c:canAccess>
       <exists var="rep">
          <w3c:acMember>#rep</w3c:acMember>
          <w3c:employee>#rep</w3c:employee>
       </exists>
    </then>
  </if>
</forall>
```

上記の意味は：全ての X について、もし、X が W3C のメンバーであれば、X はメンバーページにアクセスすることができ、誰か代表(rep)がいて、顧問委員会(Advisory Committee：AC)の X 代表の委員で、かつ X の従業員である。これは数学の表記に比べれば汚いが、XML としてはそれほどでもない。

var 属性が変数を ID 空間（URI 空間のサブセット）で定義している、XML の ID 型を要素の識別子として用いるために、IDREF 型を持たなければならない。（替わりに、XML のエンティティを用いて、新しい魔法のエンティティ &x を導入したり、ドル記号にうんざりするかもしれないが単純に新しい構文として $x を変数としたりすることもできる、あるいは、もし名前空間を作るのが好きなら、変数からなる名前空間を作っても良い。最後のものは、エンジンがそれを理解できないと混乱するだろう。）

（XML の名前空間はスコープを用いていないが、”forall”節は節のスコープ内でしか意味を持たない変数の導入が必要であることに注意しよう。しかし、置換(substitution)が定義される時に、外から参照されてしまうかもしれない。変数が first class object だと便利になるので、例えば、「規則 foo.rdf#rule1 の中の変数 foo.rdf#name の値を "John Doe"に置き換えたい」と思うだろう。一つの式で、限量記号を用いた式を２つ以上用いる時は、勿論注意が必要である。）

1.0 版の構文仕様では、quantification を用いた特定の論理式のための特別な構文が用意されていた。

```
<rdf:Description aboutEach="#pages">

  <s:Creator>Ora Lassila</s:Creator>

</rdf:Description>
```

これは、今なら以下の意味であると記述できる。

Note この点を指摘してくれた Dan Connolly に感謝する。

```
<forall var="x">
 <if>
   <rdf:li for="#pages" value="#x">
   <then>
      <s:Creator for="#x">Ora Lassila</s:Creator>
   </then>
 </if>
</forall>
```

Definitive lists

definitive set　を用いたいことが良くある。

（例えば、definitive lists：

- ドキュメントの検証のための要求事項の definitive lists - validatable schema
- リソースのアクセス制御リスト(ACL)
- bank statement
- などなど）

W3C が W3C のメンバーのリストを与える時、そこに載っている誰かがメンバーであることを示しているだけでなく、そこに載っていない人はそうではないことを意味することができる。リストの排他性はドキュメントやドキュメントの一部にたいする statement である。以下は、the definitive nature of a list に関する statement があり、そのリストが続く：

```
<forall var="x">
  <if rdf:about="#list">
    <w3c:member id="statement"
       about="http://www.w3.org/"><var ref="#x">
    </w3c:member>
    <then>
      <implies rdf:value="#statement" />
    </then>
  </if>
</forall>
<foo:container id="list"
   rdf:about="http://www.w3.org/">
   <w3c:member>http://www.ibm.com/"</w3c:member>  訳注1）
   <w3c:member>http://www.hp.com/"</w3c:member>
   <w3c:member>http://www.netscape.com/"</w3c:member>
   <w3c:member>http://www.sun.com/"</w3c:member>
   <w3c:member>http://www.acme.com/"</w3c:member>
</foo:container>
```

通常の代数(Algebra) では、"For all"を"such that"と伴に使うが、ここでは、<forall> と <if>を用いている、その２つを組み合わせた略記として<ifany>を用いると

```
<ifany var="x" rdf:about="#list">
```

訳注1）"は不要と思われる

```
    <w3c:member id="statement"
       about="http://www.w3.org/"><var ref="#x">
    </w3c:member>
    <then>
      <implies rdf:value="#statement" />
    </then>
  </ifany>
  <foo:container id="list"
     rdf:about="http://www.w3.org/">
     <w3c:member>http://www.ibm.com/"</w3c:member>
     <w3c:member>http://www.hp.com/"</w3c:member>
     <w3c:member>http://www.netscape.com/"</w3c:member>
     <w3c:member>http://www.sun.com/"</w3c:member>
     <w3c:member>http://www.acme.com/"</w3c:member>
  </foo:container>
```

これは、これまで定義してきた機能で行える。（訳注：詳細な注は略）

関数（Functions）

関数はパラメータを抽出し意味をカプセル化(encapsulate)する能力である。関数は、実際の言語に多くの関数の形式があるように、RDF と XML にもいくつかの方法で対応させることができる。

データの表現をみると、関数は共通の表現のコンパクトな表現になる。簡略表現でも幾つかの形式がある（位置による引数や名前つきパラメータ）が、RDF で書くのであれば、引数[関数起動時に仮パラメータを置き換えるもの]をプロパティノードで表現した RDF ノードがとするのが自然だろう。そのノードから導かれる情報の集合は、関数の本体(”body”)に対応する。みんなある関数の”the”意味を関数定義者が表現する推論規則で考えるが、他のドキュメントでは独自に規則を追加するかもしれないのが、Semantic Web の哲学の面白いところだ。言い換えると関数本体(function body)は、それほど便利な用語ではなく、関数に関す任意の表現が同じ事ができる。例えば、前に出てきた例題

```
  <forall id="baz" var="x" rdf:about="#x">
    <if>
      <w3c:memberOf>http://www.w3.org/</w3c:memberOf>
      <then>
         <w3c:canAccess>http://www.w3.org/Member</w3c:canAccess>
         <exists var="rep">
            <w3c:acMember>#rep</w3c:acMember>
            <w3c:employee>#rep</w3c:acMember>
         </exists>
      </then>
    </if>
  </forall>
```

が良い例である。これは W3C のメンバーシップの概念についての幾つかの情報を述べている。これを definitive にしても良いが、言語というよりは信頼モデルの一部である。言い換えると、W3C は、W3C のメンバーならば、従業員を AC 代表として持つという。他の人は、W3C に入るような組織は、誰かしら賢いやつが一人位いるものだと思うかもしれない。

特定の RDF ノードは作り手により特定の事象を意図する、すくなくともスキーマはそれら事象へのポインタを持っていると私は思っている。
上記の例では、推論は、メンバーシップのプロパティ：プロパティは二引数の述語(binary predicate)として用いられているが、複数のパラメータを持つより一般の n 引数(n-ary)の式でも同様に：

```
<forall var="x" v2="y" v3="z" rdf:about="#x">
  <if>
    <employee>
      <name>#y</name>
      <street>#s</street>
      <zip>#z</zip>
    <employee>
    <then>
       [...]
    </then>
  </if>
</forall>
```

基本(basic)RDF ユーティリティは、全ての種類の形式を許すが、いくつかの共通の形式を取り上げた法が便利である。上記の例では、この規則はは、名前(name)とストリート(street)（訳注：と郵便番号(zip)）を持つすべての（どこのでもよい）従業員(employee)のノードに適用される。”employee”というプロパティ名は、関数名のように用いられている。替わりに以下のように記述することもできる。

```
<forall var="x" v2="y" v3="z" rdf:about="#x">
  <if>
    <rdf:type>http://www.w3.org/1999/a/empType</>
    <z:name>#y</>
    <z:street>#s</>
    <z:zip>#z</>
    <then>
       [...]
    </then>
  </if>
</forall>
```

ここでは、empType で明示的に与えら、与えられたパラメータを持つノードに対して規則が適用される。勿論、こらの２つはあ、RDF スキーマタイププロパティでリンクされる。

```
<rdfs:range about="#employer">http://www.w3.org/1999/a/empType</a>

<forall var="x" v2="y" rdf:about="#x">
  <if>
    <employer>#y</employer>
    <then>
      <rdf:type>http://www.w3.org/1999/a/empType</rdf:type>
    </then>
  </if>
</forall>
```

ここでは関数について議論しているのだが、RDF スキーマの仕様について少し定義しておく。まずは、プロパティの”range”とは：

```
<forall var="aPropertyName" v2="y" v3="aType" rdf:about="#x">
  <if>
    <rdfs:range about="#aPropertyName">#aType</a>
    <then>
      <if>
        <#aPropertyName>#y</>     <!-- oops!  ->
      <then>
        <rdf:type about="#y">http://www.w3.org/1999/a/empType</rdf:type>
      </then>
    </then>
  </if>
</forall>
```

RDF 宣言が full ID でなされなければならないことは判っている。これは識別子の問題である。

```
<#aPropertyName>#y</>     <!-- oops!  ->
```

は必要なのだけれど、XML では書けないので、XML 要素を用いて、

```
<rdf:property pname="#aPropertyName">#y</>     <!-- better!  ->
```

とすれば、言語の一貫性の観点からはいまいち綺麗ではないが、XML としては許せる。@@@

スコーレム関数（Skolem functions）

全ての人には、母があり、人の母は一意に決まることは知られているので、この場合、x は a の母親であるような任意の x と書く替わりに、a の母親として参照した方が簡単である。（同様にスコーレム関数は、記号論理から限量記号を消去するために使われる）

XML の略記は：

```
<the pname="#mother" of="#a">
```

これは質問(query)と考えることもできる。これは、プロパティが一意の値を持つときに定義されるが、一意でないときには、どのような暗黙の quantification が用いられどのようなスコープを持つのか明確でなくなる。

２つの選択肢がある；

母親の一意性の宣言を含意するフレーズを表現の中に入れ込む。

F(..., THE(prop,x)...) -> (exists(w). prop(w,x)) & ((prop(x,y) & prop(z,y)) -> x=z).

ここで、大文字の THE(prop)はスコーレム化のための特殊な関数であり、THE (prop)は追加により言語を新しくする新しい関数である。THE は、述語を引数に取るため１階の関数ではない。存在文から生成されるスコーレム関数のような関数ですらない。

1. 存在の宣言と等価だが、一意性の宣言とは異なる。

   F(.., A(prop,x)..) -> (exists(w. prop(w,x)).

後者はスコーレム化のために使われており、我々はそれに固執すべきだと考える。否定の関数を考えてみよう。それは言語の一部ではなく、ショートカットであった。

証明（Proof）

これは証明の constructing ではないが、検証の証明のための(transmitting)に関するものである。証明言語を

定義するために、proof checking machine のパワーを定義する必要がある。言い換えると、原子的な細かなステップで、過保護に面倒を見なければならないのか、なんらかの一気に行う飛躍があるのか？この選択は基本的なものではないのだけれど、例えば、異なる能力の異なるエンジンを比較するときに語彙も異なっていることをイメージできるだろう。極端なもので、全ての事象を二項演算子の標準形(canonical form)で書かなければならないような単純な logical engine がある。もうひとつの極端な例は、"A follows-indirectly-from B" の証明（bounded だが、証明とは呼ばない）を外延の検索(extensive search)で行うものがある。この間に工学的に妥当なものがあるだろう。以下は厳密な派生ではないが、

1. 標準化(canonicalization):
2. 抽出(Extraction):
3. 置換(Substitution):
4. 含意(Implication):
5. 差演算(Dereference):
6. などなど

Follows-from

意味：情報 A の全ては情報 B から導き出される

コメント：これは、"oh, yeah?" button のためのツール。これにより、宣言からその元になった宣言を辿ることができる。宣言を検証するために、ヒントのために A だけを破棄して、同じ意味を抽出するのに、B がパースされれば B が検証される。B が書かれているとか、B を信じるべき理由などの表現は、言語には存在しない。

例：

```
<a:record id="foo" about="http://ww.w3.org/">
    <w3c:member>http://www.ibm.com/</w3c:member>
</a:record>
<rdf:description about="#foo">
    <follows-from>http://www.w3.org/MemberList</follows-fromsource>
</rdf:description>
```

この宣言文では、IBM は W4C のメンバーであることは W3C のメンバーリストから導かれる。（ドキュメントは、ドキュメントから導出されたものを宣言できるか？宣言は宣言なので、形式的にはもちろんできる。しかし、もしそのドキュメントを信用していない場合は、例えば、与えられた URI をチェックするよう要請されていかのように扱うだろう。後者では、期限切れにとして扱ったり、、“ xxx の中に yyy があるのを見つけたが、どうも信用できない。調べてくれない ” などの明示的な信用が得られない文があるかのように扱う。）

詳細な導出構文 *Specific derivation syntax*

…@@@@

電子署名と信用（*Digital Signature and Trust****）***

上記は論理を扱っている／現実世界や Web 上で行われている任意の推論は、信用の規則に従ってなされている。Web 上では、信用は、公開鍵暗号の力、特に電子署名を活用する。W3X の Signed XML の活動は、公開鍵に対応する私的鍵により署名されたことを示すことができる、XML ドキュメントに署名方法を定義している。以下は、強力で一般的と思われる信用に関するモデルである。基本的な考え方は、鍵の集合を用いた "assured by"文を用いることである。これは新しい言葉だが、もっと良い言葉があれば教えて欲しい。この文

は、鍵で検証される署名の付いたドキュメント（やドキュメントの一部）でも、それらから論理的に導かれるものにも適用することができる。異なる鍵集合で保証(assured by)された文の組み合わせから論理的に導出される場合には、それらの鍵の和集合で保証(assured by)される。（以下略）

結論

XML は、明らかに形式論理と信用を表現する（酷い、良い）方法である。

付録：位相(topology)に関する宣言

ここには宣言に関する宣言を思いつくままあげる。

このリストの中で、意味論の説明（例題）では A に関する宣言があり、その在るプロパティの値が B である場合の説明をしている。

含意（A Implies B）

意味：プロパティタイプ A に対する任意の宣言は、同じ subject node と値をプロパティ B についても宣言されていることを意味する。

コメント：

領域とレンジ：Subject と Object は両方とも RDF の宣言で識別されてけければならない。

例：<implies rdf:about="#from" rdf:value="#responsible">

もし、A が B から来た(“ from ” B)なら、A は responsible の値も B である。

逆（A Inverse B）

意味：プロパティタイプ A の任意の宣言は、プロパティタイプ B の逆方向の宣言を含意している。つまり、（ある Subject の）A の値であるものの B の値はその Subject となる。

コメント：

領域とレンジ：Subject と Object は RDF の宣言の中で識別されている必要がある。ある関係は、self-inverse である。逆自身も self-inverse である。

例：<implies rdf:about="#part-of" rdf:value="#includes">

もし、A が B の一部(A is “part-of” B)なら、B は A を含む(B “includes” A)[訳注]

Acyclic, etc

...@@@

[訳注] 原文は、A "includes" B　だが…

まとめ：導入された項（Terms）（Terms introduced - A summary）

Toolbox terms			
項	**言語上の役割**	**axiomatic status**	**意味**
rdf:about	xml attribute	syntactic sugar	エレメントの中身に rdf のデフォルトの Subject をセットする
rdf:for	xml attribute	syntactic sugar	rdf subject は this element で書き換えられる
head	xml element	syntactic sugar	set default rdf subject to parent element's node
not	xml element	fundamental addition	implies (reifiation and) denial of contents
truth	rdf property	strict alternative to *not*	ドキュメントの一部に対する真偽値の宣言
If	xml element	synonym sugar	条件文を構成するための not の同意語
then	xml element	syntactic sugar	条件文を構成するための not の同意語
forall	xml element	fundamental	（全称）限量子
exists	xml element	syntactic sugar	there exists - not(forall(not ...))から導かれる

参考文献 References

these always seem to diappear... theer are many small lists of these, all different.
Ban logic@@
Appel'set al. work at Princeton on Proof-Carrying Authentication: Proof-Carrying Authentication. Andrew W. Appel and Edward W. Felten, 6th ACM Conference on Computer and Communications Security, November 1999.

最終更新 $Id: blank.html,v 1.1 1999/05/24 14:24:19 timbl Exp $

Tim BL