TRIPLE—An RDF Query, Inference, and Transformation Language

Michael Sintek
DFKI GmbH
Kaiserslautern
sintek@dfki.de

Stefan Decker
Stanford University
Database Group
stefan@db.stanford.edu

Abstract

This paper presents TRIPLE, a layered and modular rule language for the semantic web [1]. TRIPLE is based on Horn logic and borrows many basic features from F-Logic [9] but is especially designed for querying and transforming RDF models [17].

TRIPLE can be viewed as a successor of SiLRI (Simple Logic-based RDF Interpreter [5]). One of the most important differences to F-Logic and SiLRI is that TRIPLE does not have a fixed semantics for object-oriented features like classes and inheritance. Its layered architecture allows such features to be easily defined for different object-oriented and other RDF extensions like RDF Schema [16]. Description logics extensions of RDF (Schema) like OIL [14] and DAML+OIL [3] that cannot be handled directly by Horn logic are provided as modules that interact with a description logic classifier, e.g. FaCT [8], resulting in a hybrid rule language. This paper sketches syntax and semantics of TRIPLE.

Keywords: Semantic Web, RDF, DAML, Logic Programming, F-Logic

1 Introduction

TRIPLE is a layered rule language, aiming to support applications in need of RDF reasoning and transformation. The core language is based on Horn logic which is syntactically extended to support RDF primitives like namespaces, resources, and statements (triples, which gave TRIPLE its name). This core language can be compiled into Horn logic programs and enacted by Prolog systems like XSB [15].

Inference systems for higher level-languages like RDF Schema and DAML+OIL can either be implemented directly in TRIPLE or are provided as modules interacting with external reasoning components.

TRIPLE provides a (human readable) Prolog-like syntax (both in mathematical and ASCII notation; cf. appendix A) as well as an RDF-based syntax.

This work was supported by the German Ministry for Education and Research, bmb+f (Grant: 01 IW 901, Project FRODO: A Framework for Distributed Organizational Memories).

In this section we introduce TRIPLE (using its mathematical Prolog-like notation). Section 2 presents the layered architecture of TRIPLE, Section 3 introduces its RDF-based syntax (for the subset TRIPLE_0), and Section 4 gives a semantic characterization. Section 5 finally concludes the paper.

The reader is supposed to be familiar with RDF and RDF Schema.

1.1 Features of TRIPLE

In the following, the main features of TRIPLE (i.e., those extending Horn logic) are informally described. Note that not all the features are available in TRIPLE_0 (cf. Section 2).

Namespaces and Resources TRIPLE has special support for namespaces and resource identifiers. Namespaces are declared via clause-like constructs of the form *nsabbrev* := *namespace*., e.g.

rdf := "http://www.w3.org/1999/02/22-rdf-syntax-ns#".

Resources are written as *nsabbrev:name*, where *nsabbrev* is a namespace abbreviation and *name* is the local name of the resource.

Resource abbreviations can be introduced analogously to namespace abbreviations, e.g.

$$\text{isa} := \text{rdfs:subClassOf}.$$

Statements and Molecules An RDF statement (triple) is—inspired by F-Logic object syntax—written as

$$\textit{subject}[\textit{predicate} \rightarrow \textit{object}]$$

Several statements with the same subject can be abbreviated as "molecules":

$$\text{stefan}[\text{hasAge} \rightarrow 33; \text{isMarried} \rightarrow \text{yes}; \ldots]$$

RDF statements (and molecules) can be nested, eg.:

$$\text{stefan}[\text{marriedTo} \rightarrow \text{birgit}[\text{hasAge} \rightarrow 32]]$$

Models RDF models, i.e., sets of statements, are made explicit in TRIPLE ("first class citizens").[1] Statements, molecules, and also Horn atoms that are true in a specific model are written as *atom*@*model* (similar to Flora-2 module syntax), where *atom* is a statement, molecule, or Horn atom and *model* is a model specification (i.e., a resource denoting a model), e.g.

$$\text{michael}[\text{hasAge} \rightarrow 34]@\text{factsAboutDFKI}$$

TRIPLE also allows Skolem functions as model specifications. Skolem functions can be used to transform one model (or several models) into a new one when used in rules (e.g., for ontology mapping/integration):

$$O[P \rightarrow Q]@\text{sf}(m1, X, Y) \;\longleftarrow\; \ldots$$

If all (or many) statements/molecules or Horn atoms in a formula (see Section 1.1) are from one model, the following abbreviation can be used: *formula*@*model*. All statements/molecules and Horn atoms in *formula* without an explicit model specification are implicitly suffixed with @*model*.

Instead of constants, variables, and Skolem functions also boolean combinations can be used, eg.: ($model_1 \cap model_2$) specifying the intersection of two models, ($model_1 \cup model_2$) specifying the union of two models, and ($model_1 \backslash model_2$) specifying the set-difference of two models.

Reified Statements Reified statements are written as <*statement*> and can be used inside other statements, allowing "modal" statements like

$$\text{stefan}[\text{believes} \rightarrow <\text{Ora}[\text{isAuthorOf} \rightarrow \text{homepage}]>]$$

Path Expressions For navigation purposes, path expressions have proven to be very useful in object oriented languages. TRIPLE allows the usage of path expressions instead of subject, predicate, or object definitions (and at all other places where terms are allowed). Path expressions are dot-delimited sequences of resources, e.g.:

$$\text{stefan.spouse.mother}$$

denotes Stefan's mother in law.

Logical Formulae TRIPLE uses the usual set of connectives and quantifiers for building formulae from statements/molecules and Horn atoms, i.e., $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, etc.[2] All variables must be introduced via quantifiers, therefore marking them is not necessary (i.e., TRIPLE does not require variables to start with an uppercase letter as in Prolog).

[1] Note that the notion of *model* in RDF does not coincide with its use in (mathematical) logics.

[2] For TRIPLE programs in plain ASCII syntax, the symbols AND, OR, NOT, FORALL, EXISTS, `<-`, `->`, etc. are used; cf. the example in Section 2.1.

Clauses and Blocks A TRIPLE clause is either a fact or a rule. Rule heads may only contain conjunctions of molecules and Horn atoms and must not contain (explicitly or implicitly) any disjunctive or negated expressions.

To assert that a set of clauses is true in a specific model, a model block is used:

$$@\mathit{model}\ \{\mathit{clauses}\}$$

or, in case the model specification is parameterized:

$$\forall\ \mathit{Mdl}\ \ @\mathit{model}(\mathit{Mdl})\ \{\mathit{clauses}\}$$

1.2 Example: Dublin Core Metadata

The Dublin Core Metadata Initiative [4] defines a set of elements for marking up documents with metadata like title, creator, date, subject, etc. An encoding of Dublin Core metadata in RDF is straightforward. The example in Figure 1 adds some simple metadata to a document and defines a (Horn) rule that searches for documents with a specified subject.[3]

```
rdf := "http://www.w3.org/...rdf-syntax-ns#".
dc := "http://purl.org/dc/elements/1.0/".
dfki := "http://www.dfki.de/".

@dfki:documents {

   dfki:d_01_01[
      dc:title → "TRIPLE";
      dc:creator → "Michael Sintek";
      dc:creator → "Stefan Decker";
      dc:subject → RDF;
      dc:subject → triples; ... ].

   ∀ S, D  search(S, D) ⟵
      D[dc:subject → S].
}
```

Figure 1: Example: Dublin Core Metadata

2 The TRIPLE Layered Architecture

As already mentioned, TRIPLE is a layered rule language. Two different kinds of layers are supported:

- syntactical extensions of Horn logic to support basic RDF constructs like resources and statements
- modules for semantic extensions of RDF like RDF Schema, OIL, and DAML+OIL, implemented either directly in TRIPLE or via interaction with external reasoning components

[3] Note that symbols in TRIPLE can be enclosed in single or double quotes; if a symbol does not contain special characters and starts with a letter, no quotes are needed. Thus, TRIPLE, 'TRIPLE', and "TRIPLE" all denote the same symbol.

TRIPLE is the extension of Horn logic as described in Section 1.1. TRIPLE_0 is the subset of TRIPLE without quantifiers and negation (and has already been implemented on top of XSB, see `http://www.dfki.uni-kl.de/frodo/triple/`), TRIPLE_0^- is the subset without quantifiers, but with negation. TRIPLE_0 and TRIPLE_0^- mainly exist to simplify the implementation of the higher layers. For TRIPLE_0, a representation in RDF exists which is explained in Section 3.

The following two sections describe the modular extensions for RDF Schema and DAML+OIL, called TRIPLE/RDFS and TRIPLE/DAML+OIL.

2.1 TRIPLE/RDFS

This section shows how rules axiomatizing (part of the) semantics of RDF Schema are implemented in TRIPLE. The rules can be used together with a Horn logic based inference engine like XSB to derive additional knowledge from an RDF Schema specification.

Figures 2 and 3 show the RDF Schema module in mathematical and plain ASCII notation.

The first lines define namespaces (for RDF and RDF Schema) and abbreviations (for type, subPropertyOf and subClassOf).

The rules are enclosed by a model specification block: $\forall\ \mathit{Mdl}\ \ @\mathrm{rdfschema}(\mathit{Mdl})\ \{\ldots\}$

The Skolem function rdfschema(Mdl) is the model identifier of all facts derived by the rules enclosed by the model specification block. The parameter Mdl denotes the RDF Schema specification. The model rdfschema(Mdl) contains all statements from the model Mdl plus everything derived additionally by the rules. The rule

$$\begin{array}{l} \forall\, O, P, V\ \ O[P \rightarrow V]\ \longleftarrow \\ \quad O[P \rightarrow V]@\mathit{Mdl}. \end{array}$$

specifies that every triple contained in the model Mdl is also element of the model with the identifier rdfschema(Mdl). The next rule defines the inheritance of values from sub properties to super properties. The remaining rules define the semantics of transitive properties (subPropertyOf and subClassOf) and of the type property.

2.2 TRIPLE/DAML+OIL

DAML+OIL [3] (and also OIL [14]) are description logics extensions of RDF Schema that cannot be mapped to Horn logic directly. For this reason, a model daml_oil(Mdl) is provided that accesses a description logics classifier (e.g., FaCT) to realize the semantics of DAML+OIL. Access to the daml_oil(Mdl) model is restricted to premises in rules; facts and rule heads must not contain any references to it.

$$\begin{array}{l}
\mathrm{rdf} := \text{"http://www.w3.org/...rdf-syntax-ns\#"}. \\
\mathrm{rdfs} := \text{"http://www.w3.org/.../PR-rdf-schema-...\#"}. \\
\mathrm{type} := \mathrm{rdf{:}type}. \\
\mathrm{subPropertyOf} := \mathrm{rdfs{:}subPropertyOf}. \\
\mathrm{subClassOf} := \mathrm{rdfs{:}subClassOf}. \\
\forall\ \mathit{Mdl}\ \ @\mathrm{rdfschema}(\mathit{Mdl})\ \{ \\
\quad \mathrm{transitive}(\mathrm{subPropertyOf}). \\
\quad \mathrm{transitive}(\mathrm{subClassOf}). \\
\quad \forall\, O, P, V\ \ O[P \rightarrow V]\ \longleftarrow \\
\qquad O[P \rightarrow V]@\mathit{Mdl}. \\
\quad \forall\, O, P, V\ \ O[P \rightarrow V]\ \longleftarrow \\
\qquad \exists\, S\ \ S[\mathrm{subPropertyOf} \rightarrow P]\ \wedge\ O[S \rightarrow V]. \\
\quad \forall\, O, P, V\ \ O[P \rightarrow V]\ \longleftarrow \\
\qquad \mathrm{transitive}(P)\ \wedge \\
\qquad \exists\, W\ \ (O[P \rightarrow W]\ \wedge\ W[P \rightarrow V]). \\
\quad \forall\, O, T\ \ O[\mathrm{type} \rightarrow T]\ \longleftarrow \\
\qquad \exists\, S\ \ (S[\mathrm{subClassOf} \rightarrow T]\ \wedge\ O[\mathrm{type} \rightarrow S]). \\
\}
\end{array}$$

Figure 2: RDF Schema in TRIPLE

```
rdf := 'http://www.w3.org/...rdf-syntax-ns#'.
rdfs := 'http://www.w3.org/.../PR-rdf-schema-...#'.
type := rdf:type.
subPropertyOf := rdfs:subPropertyOf.
subClassOf := rdfs:subClassOf.
FORALL Mdl @rdfschema(Mdl) {
  transitive(subPropertyOf).
  transitive(subClassOf).
  FORALL O,P,V   O[P->V] <-
      O[P->V]@Mdl.
  FORALL O,P,V   O[P->V] <-
    EXISTS S   S[subPropertyOf->P] AND O[S->V].
  FORALL O,P,V   O[P->V] <-
    transitive(P) AND
    EXISTS W   (O[P->W] AND W[P->V]).
  FORALL O,T   O[type->T] <-
    EXISTS S   (S[subClassOf->T] AND O[type->S]).
}
```

Figure 3: RDF Schema in TRIPLE, plain ASCII syntax

The resulting rule language is a hybrid rule language amalgamating Horn rules and description logics similar to Carin [10]. The main difference is that Carin's primary goal is to remain complete and correct. This is achieved by restricting the Horn part to function-free, recursive rules and by either restricting the description logics part by removing the constructors $\forall R.C$ and $(\leq n\, R)$ or by further restricting the Horn rules to be *role-safe* (i.e., by restricting the way in which variables can appear in role atoms in the rules, similar to safety conditions on Datalog KBs).

In TRIPLE/DAML+OIL, neither the Horn rules nor the description logics part are restricted in any way, resulting in an incomplete language. But since Prolog implementations for Horn logic are already incomplete, this does not make things worse. The resulting language is, on the other hand, quite powerful and meets the pragmatic requirements of a rule and transformation language for the semantic web.

In the DAML+OIL example in Figure 4, Herbivore and Carnivore are (incorrectly) defined to be disjoint, therefore the class Omnivore is unsatisfiable which will be revealed by the query `unsatisfiable(animals:Omnivore) @ check(animals:ontology)`.

```
daml := 'http://www.daml.org/.../daml+oil#'.
animals := 'http://www.example.org/animals#'.
@animals:ontology {
  animals:Animal[rdf:type -> daml:Class].
  animals:Herbivore[rdf:type -> daml:Class;
    daml:subClassOf -> animals:Animal].
  animals:Carnivore[rdf:type -> daml:Class;
    rdfs:subClassOf -> animals:Animal;
    daml:disjointWith -> animals:Herbivore].
  animals:Omnivore[rdf:type -> daml:Class;
    rdfs:subClassOf -> animals:Herbivore;
    rdfs:subClassOf -> animals:Carnivore].
}
FORALL Ont   @check(Ont) {
  FORALL C   unsatisfiable(C) <-
    C[daml:subClassOf ->
      daml:Nothing]@daml_oil(Ont).
}
```

Figure 4: Animals Example for TRIPLE/DAML+OIL

3 TRIPLE$_0$ in RDF

In this section, we describe how to represent TRIPLE$_0$ in RDF. Appendix B contains the RDF Schema definition for TRIPLE$_0$.

Representing a rule language like TRIPLE in RDF (or XML) allows rules to be distributed on the Web, e.g. between communicating agents, which is the primary goal of the RuleML initiative [2].

A possible scenario could be similar to that of mobile agents, e.g.: a customer intending to purchase some goods formulates his interests/preferences etc. as a set of TRIPLE rules and facts, sends them (encoded in RDF) to some vendors who enact them on their local knowledge bases (after transformation into their own rule languages), and then send the results back to the buyer.

Namespace for TRIPLE in RDF In the following, 'triple' denotes the TRIPLE namespace (something like 'http://www.semanticweb.org/2001/06/30/triple#').

Abbreviations Abbreviations for namespaces and resources are not necessary: we simply use the XML namespace and entity declarations.

Triples, Molecules, Path Expressions $a[b \rightarrow c]$ becomes an instance of triple:Triple which is a subclass of rdf:Statement:

```
<triple:Triple>
  <triple:subject rdf:resource="#a"/>
  <triple:predicate rdf:resource="#b"/>
  <triple:object rdf:resource="#c"/>
</triple:Triple>
```

There is no need for an RDF representation of molecules like $a[b \rightarrow c; p \rightarrow q; \ldots]$ since they are equivalent to the conjunction of single Triples. The same holds for path expressions (which can be split into separate Triples).

Associated Models, Model Expressions Every Triple can have an *associated* model: $a[b \rightarrow c]@m$ becomes

```
<triple:Triple>
  <triple:subject rdf:resource="#a"/>
  <triple:predicate rdf:resource="#b"/>
  <triple:object rdf:resource="#c"/>
  <triple:model rdf:resource="#m"/>
</triple:Triple>
```

Note that triple:model is a property that may be used on all formulas and clauses, not only on Triples (see the section on @-Expressions below). Any term can be used as a model; complex model expressions can be built with triple:ModelUnion, triple:ModelIntersection etc., e.g.:

```
<triple:ModelUnion>
  <triple:firstModel rdf:resource="#m"/>
  <triple:secondModel rdf:resource="#n"/>
</triple:ModelUnion>
```

Furthermore, a triple model may be denoted by a Skolem function to allow parameterized models (triple:SkolemModel).

Terms triple:Term comprises rdfs:Literal, triple:Variable, triple:Structure, triple:Resource, triple:ReifiedTriple, triple:Model etc.

Atoms and Formulas We have two sorts of Atoms: triple:Triple and triple:HornAtom, where HornAtoms are the normal Horn atoms like p(a,X).

Since we do not support Lloyd-Topor transformations in TRIPLE$_0$, Atom and And/Or formulas are the only formulas.

$$
\begin{aligned}
A : N &\longrightarrow \text{resource}(A, N) && (1)\\
O[P \to V] &\longrightarrow \text{statement}(O, P, V) && (2)\\
S@M &\longrightarrow \text{true}(S, M) \quad \text{for statements (and atoms) } S && (3)\\
<S> &\longrightarrow S \quad \text{for statements } S && (4)\\
O[P_1 \to V_1; P_2 \to V_2; \ldots]@M &\longrightarrow O[P_1 \to V_1]@M \ \wedge && (5)\\
& O[P_2 \to V_2]@M \ \wedge \ \ldots &&\\
\text{true}(S, M_1 \cap M_2) &\longrightarrow \text{true}(S, M_1) \ \wedge \ \text{true}(S, M_2) && (6)\\
\text{true}(S, M_1 \backslash M_2) &\longrightarrow \text{true}(S, M_1) \ \wedge \ \neg \ \text{true}(S, M_2) && (7)\\
X := Y.\ S(X) &\longrightarrow \forall\ X \ \ (X = Y \ \wedge \ S(X)) && (8)\\
& \text{for clause sequences } S(X) &&
\end{aligned}
$$

Figure 5: The RDF-specific Rewrite Rules

Clauses A triple:Clause simply consists of a head (with range triple:Atom) and a body (with range triple:Formula), both of which may be empty to form facts and queries. It may also have an associated model (see below).

@-Expressions All forms of @-expressions are mapped to usages of the triple:model property, even for the { } enclosed blocks, e.g.

```
@someModel {
  clause1.
  clause2.
}
```

becomes

```
<triple:Clause rdf:ID"clause1">
  <triple:model rdf:resource="#someModel"/>
</triple:Clause>

<triple:Clause rdf:ID"clause2">
  <triple:model rdf:resource="#someModel"/>
</triple:Clause>
```

4 Semantic Characterization of TRIPLE

This section provides a first indirect semantic characterization of TRIPLE by defining a mapping to Horn Logic. This allows TRIPLE to be implemented on top of XSB (i.e., Prolog with tabled resolution), analogously to the F-Logic Flora [12].

Figure 5 shows the rewrite rules for mapping RDF-specific features like resources and statements. All other mappings are well-known (Lloyd-Topor transformations for handling of quantifier [11]) or straightforward (see the SiLRI system [5]). Example:

p:jdow[p:lastname $\to$ doe]@$m1$. $\longrightarrow$
`true(statement(resource(p, jdow), resource(p, lastname), doe), m1).`

In a future document, a model-theoretic semantics based on minimal Herbrand models and fix-point operators will be provided.

5 Conclusion

In this paper, we presented TRIPLE, a novel query and transformation language for RDF. Its core is a syntactical extension of Horn logic similar to F-Logic, but specialized for the requirements on the semantic web by making web resources, (RDF) models, and statements first class citizens.

Its main purpose is to query web resources in a declarative way, e.g. for intelligent information retrieval based on background knowledge like ontologies and search heuristics. For early approaches in this area, refer to, e.g., [7, 6, 13].

TRIPLE's layered architecture allows extensions of RDF to be implemented as extension modules (via parameterized models). Simple object-oriented extensions like RDF Schema can be directly implemented with the extended Horn logic features of TRIPLE, other extensions like DAML+OIL are realized via interaction with external reasoning components like a description logics classifier.

TRIPLE's model concept (esp. the parameterized models) enables the transformation of models, thus enabling knowledge base and ontology mapping/integration tasks which are needed in distributed settings as the semantic web (see, e.g., [18]).

Since models are first class citizens in TRIPLE, modal functionalities as needed in agent communication are also provided (e.g., agent A "believes" statements in model M, which has been received from agent B, to be true).

TRIPLE is currently being developed by the authors. A first implementation of TRIPLE_0 based on XSB is available at: `http://www.dfki.uni-kl.de/frodo/triple/`. In this version, all RDF data and TRIPLE rules are compiled into a single PROLOG program, therefore restricting the size of the knowledge base to what the underlying PROLOG system (i.e., XSB) can handle.

Future versions will implement the complete TRIPLE language and allow querying distributed RDF data without compiling remote data to the local (PROLOG) knowledge base.

References

[1] Tim Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor.* Harper San Francisco, September 1999.

[2] Harold Boley, Said Tabet, and Gerd Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *International Semantic Web Working Symposium (SWWS)*, 2001.

[3] DAML Joint Committee. DAML+OIL, 2001. URL: `http://www.daml.org/2001/03/daml+oil-index.html`.

[4] DCMI. Dublin Core Metadata Initiative, 2001. URL: `http://purl.org/dc/`.

[5] Stefan Decker, Dan Brickley, Janne Saarela, and Jurgen Angele. A query and inference service for RDF. In *QL'98 — The Query Languages Workshop*, Boston, USA, 1998. WorldWideWeb Consortium (W3C).

[6] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al., editor, *Semantic Issues in Multimedia Systems.* Kluwer Academic Publisher, 1999.

[7] Dieter Fensel, Stefan Decker, Michael Erdmann, and Rudi Studer. Ontobroker: The Very High Idea. In *Proc. 11th Int. Florida AI Research Symposium (FLAIRS-98)*, May 1998.

[8] Ian Horrocks. The FaCT System, 2001. URL: `http://www.cs.man.ac.uk/~horrocks/FaCT/`.

[9] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, July 1995.

[10] Alon Y. Levy and Marie-Christine Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics. In *12th European Conference on Artificial Intelligence*, 1996.

[11] J.W. Lloyd and R.W. Topor. Making Prolog more Expressive. *Journal of Logic Programming*, 3:225–240, 1984.

[12] B. Ludäscher, Guizhen Yang, and Michael Kifer. FLORA: The secret of object-oriented logic programming. Technical report, SUNY at Stony Brook, 1999.

[13] Sean Luke, Lee Spector, David Rager, and Jim Hendler. Ontology-based Web Agents. In *Proceedings of First International Conference on Autonomous Agents (AA-97)*, 1997.

[14] OIL. Ontology Inference Layer, 2001. URL: `http://www.ontoknowledge.org/oil/`.

[15] SUNY. The XSB Programming System. Dept. of Computer Science, SUNY at Stony Brook, 2000. URL: `http://www.cs.sunysb.edu/~sbprolog/xsb-page.html`.

[16] W3C. Resource Description Framework (RDF) Schema Specification 1.0, 2001. URL: `http://www.w3.org/TR/2000/CR-rdf-schema-20000327/`.

[17] W3C. Semantic Web Activity: Resource Description Framework (RDF), 2001. URL: `http://www.w3.org/RDF/`.

[18] Gio Wiederhold, editor. *Intelligent Integration of Information.* Kluwer Academic Publishers, July 1996.

A BNF for TRIPLE

The following BNF for TRIPLE was automatically generated with the jjdoc tool which is part of SUN's JavaCC compiler generator.

```
          Program::= ( ClauseSeq <EOF> )
      ClauseBlock::= ( ( ( ForallQuantifier )? <AT> StructTerm )? SimpleClauseBlock )
SimpleClauseBlock::= ( "{" ClauseSeq "}" )
        ClauseSeq::= ( Clause ( ClauseSeq )? )
           Clause::= ( ClauseBlock | ( ( ForallQuantifier )? Term <EOC> ) )
 ForallQuantifier::= ( <FORALL> IdTermSeq )
             Term::= Op1200Term
       Op1200Term::= ( ( ( Unop1200 )? Op1100Term ) ( Binop1200 Op1200Term )? )
       Op1100Term::= ( Op1000Term ( Binop1100 Op1100Term )? )
       Op1000Term::= ( Op900Term ( Binop1000 Op1000Term )? )
        Op900Term::= ( ( Unop900 Op900Term ) | ( Quantop900 IdTermSeq
                     Op900Term ) | Binop900Term )
     Binop900Term::= ( Op700Term ( Binop900 Binop900Term )? )
        Op700Term::= ( Op680Term ( Binop700 Op700Term )? )
        Op680Term::= ( Op661Term ( Binop680 Op680Term )? )
        Op661Term::= ( Op500Term ( Binop661 Op661Term )? )
        Op500Term::= ( ( ( Unop500 )? Op400Term ) ( Binop500 Op500Term )? )
        Op400Term::= ( StructTerm ( Binop400 Op400Term )? )
       StructTerm::= ( UnitTerm ( ArgList | SBArgList )* )
         UnitTerm::= ( IdTerm | Integer | "(" Term ")" | "<" Term ">" |
                     SimpleClauseBlock )
           IdTerm::= ( ( Variable | Symbol ) ( <COLON> IdTerm )? )
         Variable::= ( "?" <SYMBOL> )
           Symbol::= ( <SYMBOL> | <Q_SYMBOL> | <DQ_SYMBOL> )
          Integer::= ( <INTEGER_LITERAL> )
          ArgList::= ( "(" TermSeq ")" )
        SBArgList::= ( "[" TermSeq "]" )
          TermSeq::= ( Op900Term ( ( <COMMA> | <SEMICOLON> ) Op900Term )* )
        IdTermSeq::= ( IdTerm ( <COMMA> IdTerm )* )
         Unop1200::= ( <IMPLIEDBY> )
        Binop1200::= ( <IMPLIEDBY> | <EQUIV> | <ASSIGN> )
        Binop1100::= ( <SEMICOLON> | <OR> )
        Binop1000::= ( <COMMA> | <AND> )
       Quantop900::= ( <FORALL> | <EXISTS> )
          Unop900::= ( <NOT> | <NEG> )
         Binop900::= ( <IMPLIES> )
         Binop700::= ( <EQUALS> | <IS> )
         Binop680::= ( <AT> )
         Binop661::= ( <DOT> )
          Unop500::= ( <PLUS> | <MINUS> )
         Binop500::= ( <PLUS> | <MINUS> )
         Binop400::= ( <TIMES> | <BY> | <INTERSECT> | <UNION> | <DIFF> )
```

B RDF Schema for TRIPLE_0

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
 <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
 <!ENTITY triple 'http://www.semanticweb.org/2001/06/30/triple#'> ]>
<rdf:RDF xmlns:rdf="&rdf;"  xmlns:rdfs="&rdfs;"
 xmlns:triple="&triple;"  xmlns="&triple;">

<rdfs:Class rdf:ID="Triple">
  <rdfs:subClassOf rdf:resource="&rdf;Statement"/>
  <rdfs:subClassOf rdf:resource="&triple;Atom"/>
</rdfs:Class>

<rdf:Property rdf:ID="subject">
  <rdfs:subPropertyOf rdf:resource="&rdf;subject"/>
  <rdfs:domain rdf:resource="&triple;Triple"/>
  <rdfs:range rdf:resource="&triple;Term"/>
</rdf:Property>

<rdf:Property rdf:ID="predicate">
  <rdfs:subPropertyOf rdf:resource="&rdf;predicate"/>
  <rdfs:domain rdf:resource="&triple;Triple"/>
  <rdfs:range rdf:resource="&triple;Term"/>
</rdf:Property>

<rdf:Property rdf:ID="object">
  <rdfs:subPropertyOf rdf:resource="&rdf;object"/>
  <rdfs:domain rdf:resource="&triple;Triple"/>
  <rdfs:range rdf:resource="&triple;Term"/>
</rdf:Property>

<rdfs:Class rdf:ID="Model">
  <rdfs:subClassOf rdf:resource="&triple;Term"/>
</rdfs:Class>

<rdfs:Class rdf:ID="SimpelModel">
  <rdfs:subClassOf rdf:resource="&triple;Model"/>
</rdfs:Class>

<rdfs:Class rdf:ID="SkolemModel">
  <rdfs:subClassOf rdf:resource="&triple;Model"/>
</rdfs:Class>

<rdf:Property rdf:ID="skolemFunction">
  <rdfs:domain rdf:resource="&triple;SkolemModel"/>
  <rdfs:range rdf:resource="&triple;Structure"/>
</rdf:Property>

<rdfs:Class rdf:ID="BinaryModelExpression">
  <rdfs:subClassOf rdf:resource="&triple;Model"/>
</rdfs:Class>

<rdf:Property rdf:ID="firstModel">
  <rdfs:domain rdf:resource="&triple;BinaryModelExpression"/>
  <rdfs:range rdf:resource="&triple;Model"/>  <!-- Term ? -->
</rdf:Property>

<rdf:Property rdf:ID="secondModel">
  <rdfs:domain rdf:resource="&triple;BinaryModelExpression"/>
```

```xml
  <rdfs:range rdf:resource="&triple;Model"/>
</rdf:Property>

<rdfs:Class rdf:ID="ModelIntersection">
  <rdfs:subClassOf rdf:resource="&triple;BinaryModelExpression"/>
</rdfs:Class>

...

<rdfs:Class rdf:ID="Term"/>

<rdfs:Class rdf:ID="Variable">
  <rdfs:subClassOf rdf:resource="&triple;Term"/>
</rdfs:Class>

<Description rdf:about="&rdfs;Literal">
  <rdfs:subClassOf rdf:resource="&triple;Term"/>
</Description>

<rdfs:Class rdf:ID="Resource">
  <rdfs:subClassOf rdf:resource="&triple;Term"/>
</rdfs:Class>

<rdfs:Class rdf:ID="ReifiedTriple">
  <rdfs:subClassOf rdf:resource="&triple;Term"/>
</rdfs:Class>

<rdf:Property rdf:ID="triple">
  <rdfs:domain rdf:resource="&triple;ReifiedTriple"/>
  <rdfs:range rdf:resource="&triple;Triple"/>
</rdf:Property>

<rdfs:Class rdf:ID="Structure">
  <rdfs:subClassOf rdf:resource="&triple;Term"/>
</rdfs:Class>

<rdf:Property rdf:ID="functor">
  <rdfs:domain rdf:resource="&triple;Structure"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="args">
  <rdfs:domain rdf:resource="&triple;Structure"/>
  <rdfs:range rdf:resource="&triple;TermSeq"/>
</rdf:Property>

<rdfs:Class rdf:ID="TermSeq">
  <rdfs:subClassOf rdf:resource="&rdf;Seq"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Formula"/>

<rdf:Property rdf:ID="model">
  <rdfs:domain rdf:resource="&triple;Clause"/>
  <rdfs:domain rdf:resource="&triple;Formula"/>
  <rdfs:range rdf:resource="&triple;Term"/>
</rdf:Property>

<rdfs:Class rdf:ID="BinaryFormula">
  <rdfs:subClassOf rdf:resource="&triple;Formula"/>
</rdfs:Class>
```

```
<rdf:Property rdf:ID="firstFormula">
  <rdfs:domain rdf:resource="&triple;BinaryFormula"/>
  <rdfs:range rdf:resource="&triple;Formula"/>
</rdf:Property>

<rdf:Property rdf:ID="secondFormula">
  <rdfs:domain rdf:resource="&triple;BinaryFormula"/>
  <rdfs:range rdf:resource="&triple;Formula"/>
</rdf:Property>

<rdfs:Class rdf:ID="And">
  <rdfs:subClassOf rdf:resource="&triple;BinaryFormula"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Or">
  <rdfs:subClassOf rdf:resource="&triple;BinaryFormula"/>
</rdfs:Class>

<rdfs:Class rdf:ID="UnaryFormula">
  <rdfs:subClassOf rdf:resource="&triple;Formula"/>
</rdfs:Class>

<rdf:Property rdf:ID="formula">
  <rdfs:domain rdf:resource="&triple;UnaryFormula"/>
  <rdfs:range rdf:resource="&triple;Formula"/>
</rdf:Property>

<rdfs:Class rdf:ID="Atom">
  <rdfs:subClassOf rdf:resource="&triple;Formula"/>
</rdfs:Class>

<rdfs:Class rdf:ID="HornAtom">
  <rdfs:subClassOf rdf:resource="&triple;Atom"/>
</rdfs:Class>

<rdf:Property rdf:ID="predicateSymbol">
  <rdfs:domain rdf:resource="&triple;HornAtom"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="#args">
  <rdfs:domain rdf:resource="&triple;HornAtom"/>
</rdf:Property>

<rdfs:Class rdf:ID="Clause"/>

<rdf:Property rdf:ID="head">
  <rdfs:domain rdf:resource="&triple;Clause"/>
  <rdfs:range rdf:resource="&triple;Atom"/>
</rdf:Property>

<rdf:Property rdf:ID="body">
  <rdfs:domain rdf:resource="&triple;Clause"/>
  <rdfs:range rdf:resource="&triple;Formula"/>
</rdf:Property>

</rdf:RDF>
```