# Title: An attempt at a formal semantics for SPARQL

Author: Fred Zemke
Date: June 14, 2006

## Abstract

This paper attempts to clarify the formal semantics of SPARQL. It is written as an explicit change proposal for clarity, but this should only be taken as a strawman proposal at this time.

## References

[SPARQL] "SPARQL query language for RDF", Candidate Recommendation, http://www.w3.org/TR/2006/CR-rdf-sparql-query-20060406/

## 1. Introduction

This paper attempts to clarify the formal semantics of SPARQL. It is written as an explicit change proposal for clarity, but this should only be taken as a strawman proposal at this time.

## 2. Proposal conventions

This proposal uses the following conventions:

| | |
|---|---|
| 1. SMALLCAPS | denote numbered editorial instructions; |
| ~~blue strikeout~~ | denotes existing text to be deleted; |
| **bold red** | denotes new text to be inserted; |
| plain | denotes existing text to be retained; |
| *[Note:...]* | brackets enclose italicized notes to the proposal reader; |

## 3. An attempt at a formal semantics

### 3.1 Changes to Table of contents

1. INSURE THAT THE NAME OF EACH SECTION IN THE TABLE OF CONTENTS IS THE SAME AS THE NAME IN THE TEXT BODY. CHANGES DUE TO THIS PROPOSAL ARE NOTED HERE:

2.1.4 Syntax for blank ~~nodes~~ **node identifiers**

2.8.3 ~~Blank nodes in query results~~ **Syntactic equivalents of blank node identifiers**

## 3.2 Changes to 2.1.4, Syntax for blank nodes

1. CHANGE THE TITLE TO

   Syntax for blank node **identifiers**

2. CHANGE THE TEXT TO READ:

   A blank node **identifier** can appear in a query pattern ~~and will take part in the pattern matching~~. Blank ~~nodes~~ **node identifiers** are indicated by ~~either~~ the form "_:a" ~~or use of "[ ]"~~. **Each distinct blank node identifier denotes a distinct blank node, which is also distinct from all nodes of the dataset. The formal semantics of SPARQL uses the function BlankNode(), which maps each blank node identifier to the blank node denoted by that blank node identifier, and anything else to itself.** Further syntactic forms involving blank nodes are described below.

   **Semantically, a blank node is treated as if it were an existentially quantified variable (using "variable" in the sense of first order predicate calculus, not in the sense of a SPARQL token matching [90] VARNAME) whose scope is a BlockOfTriples. This is a consequence of the formal semantics defined in section 2.5, "Basic graph patterns".**

   **When triple pattern matching is done using simple entailment, some people find it convenient to think of a blank node identifier as a bindable entity which may be bound to nodes of the dataset. However, when using this technique, one must note carefully that the scope of a variable is the entire query, whereas the scope of a blank node identifier is only the BlockOfTriples in which it appears. That is, using the binding semantic for blank node identifiers, a particular blank node identifier does not need to bind to the same node in the dataset when the blank node identifier appears in separate BlockOfTriples. In addition, the binding semantic may not be applicable to all entailment regimes. The more accurate model is that blank node identifiers are existentially quantified, scoped to a BlockOfTriples.**

   *[NOTE to the proposal reader: is this correct? Is the scope of a blank node identifier a BlockOfTriples, or is it a FilteredBasicGraphPattern?]*

## 3.3 Changes to 2.1.8, Result descriptions used in this document

1. EDIT THE FIRST PARAGRAPH AS FOLLOWS:

   The term "binding" is used as a descriptive term to refer to a pair of (variable, RDF term). **The term "mapping" is used as a descriptive term for a function whose**

**domain is a set of variables (possibly empty) and whose codomain is the set of RDF terms. Thus a mapping is a set of bindings such that each variable is bound at most once.**

In this document, we illustrate results in tabular form. If variable `x` is bound to `"Alice"` ~~and~~ **,** variable `y` is bound to `<http://example/x>`, **and variable z is not bound,** then we show this as:

| x | y | z |
|---|---|---|
| "Alice" | <http://example/a> | |

## 3.4 Changes to 2.2, Initial definitions

1. ADD THE FOLLOWING PREPARATORY COMMENT:

   **The term "the query" is used to indicate the SPARQL query (i.e., character string that matches the BNF for Query) of interest in a particular context. The term "the dataset" refers to the dataset that is being queried by the query.**

2. EDIT THE DEFINITION OF RDF-B AS FOLLOWS:

   let RDF-B be the set of all blank nodes in ~~RDF graphs~~ **the dataset as well as all blank nodes identified by blank node identifiers in the query**.

3. EDIT DEFINITION OF QUERY VARIABLE:

   Definition: ~~Query~~ Variable

   A ~~query~~ variable is **a VARNAME. Let V be the set of all variables.** ~~a member of the set V where V is infinite and disjoint from RDF-T.~~

   *[NOTE to the proposal reader: I have a problem with definitions that do not relate the formal semantics to the syntax. Without providing a bridge from the syntax to the semantics, we have not defined the semantics of the language.]*

4. EDIT THE DEFINITION OF SPARQL QUERY:

   A SPARQL query**, or just query for short,** is ~~a tuple (GP, DS, SM, R) where:~~

   ~~GP is a graph pattern~~

   ~~DS is an RDF Dataset~~

   ~~SM is a set of solution modifiers~~

   ~~R is a result form~~

**a character string that conforms to the BNF for [1] Query. A SPARQL query specifies the following components:**

**— a WhereClause, called the graph pattern (or, alternatively, the query pattern) of the query. As described in section 2.8 "Other syntactic forms", certain forms of syntax are abbreviations for syntax defined outside of section 2.8. It is assumed in defining the formal semantics of SPARQL that all such abbreviations have been removed from the WhereClause.**

*[NOTE to the proposal reader: do we really need two terms for one concept? I think we should standardize on either graph pattern or query pattern. Having a multiplicity of similar but distinct synonyms is potentially confusing, because the reader may think that there are two distinct meanings if the reader misses this statement.]*

**— an RDF dataset, which may be specified explicitly using one or more DatasetClauses, or implicitly to some default RDF graph, through the absence of any DatasetClause**

**— a SolutionModifier, called the solution modifiers.**

*[NOTE to the proposal reader: anyone want to align the BNF term with the plain English?]*

**— a result form, which may take one of the following forms:**

**+ 'SELECT' 'DISTINCT' ( Var+ | '*')**

**+ 'CONSTRUCT' ConstructTemplate**

**+ 'DESCRIBE' ( VarOrIRIref+ | '*')**

**+ 'ASK'**

*[NOTE to the proposal reader: perhaps we should add BNF for these possibilities, so that they can be mentioned more readily here.]*

~~The graph pattern of a query is called the query pattern.~~

## 3.5 New section 2.n, Introduction to graph pattern semantics

1. CREATE A NEW SECTION 2.N, INTRODUCTION TO GRAPH PATTERN SEMANTICS, WITH THE FOLLOWING TEXT:

**For any SPARQL query, the semantics of SPARQL defines the notion of a pattern solution (or just solution for short), and the cardinality of each solution. This section is a general introduction to the semantics of graph pattern matching.**

**A solution is a mapping (i.e., partial functions from the set of variables appearing in the SPARQL query to the set of RDF terms). A solution is said to solve the SPARQL query over the RDF dataset, using a particular E-entailment regime.**

**The cardinality of a solution is a positive integer. For many SPARQL queries, the cardinality of each solution is 1. Use of the UNION operator may produce solutions with cardinality greater than 1. One might also say that the cardinality of a mapping that is not a solution is 0, though this specification does not do so.**

**The definition of solution, and the cardinality of a solution, is distributed in various sections of this specification, effectively constituting a simultaneous recursive definition of both terms.**

## 3.6 Changes to 2.3, Triple patterns

1. REPLACE THE FORMAL DEFINITION OF TRIPLE PATTERN WITH

**After applying the expansions defined in section 2.8 "Other syntactic forms", a BlockOfTriples will conform to the following BNF:**

```
BasicGraphPattern ::=
     TriplePattern ( '.' TriplePattern )*

TriplePattern ::= Subject Verb Object

Subject ::= VarOrTerm

Verb ::= VarOrIRIref | 'a'

Object ::= VarOrTerm
```

**If TP is a character string that conforms to the BNF given above for TriplePattern, with Subj as the Subject, Vb as the Verb and Obj as the Object, then the tuple**

```
( Subj, Vb, Obj )
```

**is called the triple pattern corresponding to TP.**

**If S is a mapping, TP is a triple pattern, and every variable that appears in TP is in the domain of S, then S(TP) denotes the triple obtained by replacing each variable v in TP by S(v) and each blank node identifier in TP by the blank node that it denotes. Note that, assuming that the subject of TP is not a literal, S(TP) is an RDF graph that shares no blank node with the dataset.**

*[NOTE to the proposal reader: the extension of a mapping S to literals, IRIs and blank node identifiers is found in the next section, 2.4, Pattern solu-*

*tions. Some rearranging of the order of presentation would be good.]*

## 3.7 Changes to 2.4, Pattern solutions

1. DELETE THE FORMAL DEFINITION OF PATTERN SOLUTION.

   *[NOTE to the proposal reader: Moved to the new section 2.n, Introduction to graph pattern matching.]*

2. DELETE THE TITLE OF THIS SECTION, SINCE ALL THAT REMAINS IS AN EXAMPLE THAT MIGHT AS WELL APPEAR AT THE END OF THE PRECEDING SECTION.

## 3.8 Changes to 2.5, Basic graph patterns

1. AFTER THE DEFINITION OF BASIC GRAPH PATTERN, ADD THE FOLLOWING:

   **If P is a character string conforming to the BNF for BasicGraphPattern, then the basic graph pattern corresponding to P is the set of triple patterns corresponding to TriplePatterns contained in P. For example, the BlockOfTriples**

   ```
   [ :verb1 :obj1, ?x ] :verb2 obj3 . ?x :verb3 ?y
   ```

   **is handled by first removing all abbreviations, transforming to**

   ```
   _:a :verb1 ?x .
   _:a :verb1 :obj2 .
   _:a :verb2 :obj3 .
   ?x :verb3 ?y
   ```

   **where the query processor must effectively choose an arbitrary blank node identifier distinct from all others in the query (shown here as _:a). The BlockOfTriples shown then leads to the set**

   ```
   { ( BlankNode(_:a), :verb1, ?x ) ,
       BlankNode(_:a), :verb1, :obj2 ) ,
       BlankNode(_:a), :verb2, :obj3 ) ,
       ?x, :verb3, ?y ) }
   ```

   **which is the basic graph pattern corresponding to the original BasicGraphPattern.**

   **If S is a mapping, BGP is a basic graph pattern, and every variable that appears in BGP is in the domain of S, then we write S(BGP) to indicate the set consisting of all S(TP), where TP is a triple pattern in BGP. Note that, assuming no subject in BGP is a literal, then S(BGP) is an RDF graph that shares no blank node with the dataset.**

2. EDIT THE DEFINITION OF E-ENTAILMENT REGIME AS FOLLOWS:

   Definition: E-entailment Regime

   An E-entailment regime ~~is~~ **consists of:**

   — a ~~binary~~ relation ~~between subsets of RDF graphs~~ **whose domain is the set of all sets of RDF graphs and whose range is the set of RDF graphs**.

   *[NOTE to the proposal reader: aligns our definition with RDF Semantics.]*

   **— a function that maps each RDF graph G to a set of RDF terms, called the scoping set of the RDF graph.**

   *[NOTE to the proposal reader: replaces the non-specific language in 2.5.1 that "the scoping set may be characterized differently by different entailment regimes."]*

   ~~A graph in the range of an E-entailment is called well-formed for the E-entailment.~~

   *[NOTE to the proposal reader: not needed below.]*

3. MOVE AND EDIT THE FOLLOWING SENTENCE FROM 2.5.1 "GENERAL FRAMEWORK" TO HERE:

   The scoping set restricts the values of variable ~~assignments~~ **bindings** in a solution.

   *[NOTE to the proposal reader: might as well use the term we have already defined.]*

4. DELETE THE DEFINITION OF BASIC GRAPH PATTERN EQUIVALENCE AS FOLLOWS:

   ~~Definition: Basic graph pattern equivalence~~

   ~~Two basic graph patterns are equivalent if there is a bijection M between the terms of the triple patterns that maps blank nodes to blank nodes and maps variables, literals and IRIs to themselves, such that a triple ( s, p, o ) is in the first pattern if and only if the triple ( M(s), M(p) M(o) ) is in the second.~~

   *[NOTE to the proposal reader: no longer needed; all blank nodes denoted by blank node identifiers are already asserted to be independent of the dataset.]*

   ~~This definition extends that for RDF graph equivalence to basic graph patterns by preserving variables names across equivalent graphs~~

## 3.9 Changes to 2.5.1, General framework

1. DELETE THE DEFINITION OF SCOPING SET AND THE PARAGRAPH THAT FOLLOWS IT.

~~Definition: scoping set~~

~~A Scoping Set B is some set of RDF terms.~~

~~The scoping set restricts the values of variable assignments in a solution. The scoping set may be characterized differently by different entailment regimes.~~

*[NOTE to the proposal reader: moved into the definition of E-entailment regime.]*

2. DELETE THE DEFINITION OF SCOPING GRAPH AND THE TWO PARAGRAPHS THAT FOLLOW IT.

*[NOTE to the proposal reader: by design, the blank nodes that are denoted by blank node identifiers in the query are distinct from the blank nodes in the graph, so we don't need to construct a scoping graph with distinct blank nodes; we can just use the original graph.]*

3. EDIT THE DEFINITION OF BASIC GRAPH PATTERN E-MATCHING AS FOLLOWS:

Given an entailment regime E, a basic graph pattern BGP, **a mapping S,** and RDF graph G~~, with scoping graph G'~~, then BGP E-matches with pattern solution S on graph G ~~with respect to scoping set B~~ if:

— The domain of S includes all variables occuring in BGP.

**— There are no subjects in BGP that are literals.**

~~— BGP' is a basic graph pattern that is graph-equivalent to BGP~~

~~— G' and BGP' do not share any blank node labels.~~

~~— (G' union S(BGP')) is a well-formed RDF graph for E-entailment~~

— G E-entails (~~G'~~ **G** union S(BGP'))

— The ~~RDF terms introduced by~~ **range of** S ~~all occur in B~~ **is a subset of the scoping set of graph G under entailment regime E**.

**The cardinality of each solution of a basic graph pattern is 1.**

4. REPLACE THE LAST SENTENCE AS FOLLOWS:

~~The introduction of the basic graph pattern BGP' in the above definition makes the query basic graph pattern independent of the choice of blank node names in the basic graph pattern.~~

**Note that the blank nodes implied by blank node identifiers in the query are by design different from all blank nodes in G. There is no correspondence between blank node identifiers in a SPARQL query text and the identifiers that might be assigned to an RDF graph when that graph is presented in a**

**particular lexical format. Of course, when a solution is presented to the user, blank node identifiers may be assigned to denote blank nodes, taking care that distinct blank node identifiers denote distinct blank nodes, and such blank node identifiers may or may not be lexically equal to blank node identifiers in the SPARQL query and/or in some serialization of the dataset.**

## 3.10 Changes to 2.5.2, SPARQL basic graph pattern matching

1. EDIT THE LAST PARAGRAPH AS FOLLOWS:

A pattern solution can then be defined as follows: to match a basic graph pattern under simple entailment, it is possible to proceed by finding a ~~mapping~~ **function** from blank ~~nodes~~ **node identifiers** and variables in the basic graph pattern to terms in the graph being matched; a pattern solution is then **a mapping obtained by restricting such** a ~~mapping~~ **function** ~~restricted~~ to just the variables~~, possibly with blank nodes renamed~~. Moreover, a uniqueness property guarantees the interoperability between SPARQL systems: given a graph and a basic graph pattern, the set of all the pattern solutions is unique up to blank node renaming.

*[NOTE to the proposal reader: blank node renaming is not inherent in a mapping; it is an artefact of serializing the result to the user.]*

## 3.11 Changes to 2.5.3, Example of basic graph pattern matching

1. APPEND THE FOLLOWING MATERIAL

**Note that when using the technique of binding blank node identifiers to perform a match using simple entailment, two functions that are identical on all variables and differ only on blank node identifiers are regarded as defining a single solution. For example, given the following data**

```
<a> <b> <c1> . <a> <b> <c2>
```

**and this TriplePattern:**

```
?x <b> _:a
```

**there are two applicable functions, one that binds _:a to <c1> and one that binds _:a to <c2>. When these functions are restricted to the set of variables {?x}, there is only one mapping, which maps ?x to <a>, and so there is only one solution, of cardinality 1.**

**On the other hand, assuming the same data, but with the pattern**

```
?x <b> ?y
```

**there are two applicable functions, and when restricted to the set of variables, they are distinct, so there are two solutions, each of cardinality 1.**

## 3.12 Changes to 2.8, Other syntactic forms

*[NOTE to the proposal reader: in view of the forward references to this section, perhaps it should be moved earlier in the document.]*

## 3.13 Changes to 2.8.3 Blank nodes

1. CHANGE THE TITLE TO

**Syntactic equivalents of** Blank ~~nodes~~ **node identifiers**

## 3.14 Changes to 3.3, Value constraints - definition

1. QUESTION: DOES THIS BELONG IN SECTION 3, "WORKING WITH RDF LITERALS"? I CAN WRITE FILTER CLAUSES WITH NO LITERALS, FOR EXAMPLE FILTER (?X = ?Y).

2. REPLACE THE DEFINITION OF VALUE CONSTRAINT AS FOLLOWS:

definition: value constraint

A value constraint is a ~~boolean-valued expression of variables and RDF Terms~~ **character string that matches the BNF for Constraint**.

~~For value constraint C, a solution S matches C if S(C) is true, where S(C) is the boolean-valued expression obtained by substitution of the variables mentioned in C.~~

*[NOTE to the proposal reader: this is reformulated more accurately below.]*

3. APPEND THE FOLLOWING:

**If Q is a Constraint and S is a mapping, then section 11 "Testing values" defines how to evaluate S(Q), where S(Q) is the expression obtained by substitution of variables in Q using S. The result of such an evaluation is either an error or a boolean value.**

**If Q is a FilteredBasicGraphPattern, let $BlockOfTriples_1$, ... $BlockOfTriples_n$ be the BlockOfTriple's contained in Q, and let $Constraint_1$, ... , $Constraint_m$ be the Constraint's contained in Q.**

**Then S is a solution of Q if both of the following are true:**

**a) for all i, 1 <= i <= n, S is a solution for $BlockOfTriples_i$**

**b) for all j, 1 <= j <= m, S($Constraint_j$) is not an error and is true.**

**The cardinality of each solution of Q is 1.**

Note that blank node identifiers in a value constraint are in a different lexical scope from blank node identifiers outside the value constraint. Thus a value constraint cannot be used to filter blank nodes that may be "matched" outside the value constraint. For example, given the dataset with the single triple

```
<s> <v> 4
```

and the WhereClause

```
WHERE { ?x <v> _:a FILTER ( _:a > 3) }
```

there is one solution S to the BlockOfTriples "?x <v> _:a", which maps ?x to <s>. By the definition of basic graph pattern E-matching, this is because the dataset entails the following graph:

```
<s> <v> 4 . <s> <v> _:a
```

Now apply S to the value constraint: S ( _:a > 3 ). This is a type error because _:a denotes a blank node with no type information, which cannot be compared with the literal 3 in particular. The combined effect is that the complete WhereClause has no solutions.

Also note that two BlockOfTriple's that are separated by a Constraint are separate lexical scopes as far as blank node identifiers are concerned. For example, with the dataset

```
<s> <v1> <o1> . <s> <v2> <o2> .
```

and the WhereClause

```
WHERE { ?x <v1> _:a .
        FILTER (isIRI (?x)) .
        ?x <v2> _:a }
```

there is one solution, mapping ?x to <s>, whereas with the same dataset and the rearranged WhereClause

```
WHERE { ?x <v1> _:a .
         ?x <v2> _:a .
        FILTER (isIRI (?x)) }
```

there is no solution.

In terms of the technique to solve simple entailments by binding both variables and blank node identifiers, one must note that the bindings of blank node identifiers are not global in scope; their scope is always confined to a single BlockOfTriples.

A consequence of the definition is that the solutions of

```
SELECT ?x WHERE { }
```

**consist of all possible mappings of ?x. This might be seen as an enumeration of the scoping set, plus one additional mapping in which ?x is not bound. Similarly, the solutions of**

```
SELECT ?x ?y WHERE { }
```

**consist of all possible mappings of ?x and ?y. This might be seen as the cross product of the scoping set, extended with one additional "missing" element, with itself.**

## 3.15 Changes to 4, Graph patterns

1. APPEND THE FOLLOWING

**The root BNF for graph patterns is**

```
[20] GraphPattern ::=
     FilteredBasicGraphPattern
     ( GraphPatternNotTriples '.'? GraphPattern )?
```

**The semantics presented so far defines solution, and cardinality of each solution, for a FilteredBasicGraphPattern. These definitions are extended recursively to GraphPatterns as follows:**

**If GP is a GraphPattern that contains a FilteredBasicGraphPattern FBGP, a GraphPatternNotTriples GPNT and a (nested) GraphPattern NGP, then S is a solution for GP if S is a solution for FBGP, a solution for GPNT and a solution for NGP. The cardinality of S is determined as follows:**

**1. If FBGP is the empty pattern, let N1 be infinite; otherwise let N1 be the cardinality of S as a solution for FBGP.**

**2. Let N2 be the cardinality of S as a solution for GPNT. (Note that the grammar does not permit GPNT to be missing.)**

**3. If NGP is the empty pattern, let N3 be infinite; otherwise let N3 be the cardinality of S as a solution for NGP.**

**Then the cardinality of S as a solution for GP is the minimum of N1, N2 and N3.**

**It remains to define the semantics of a GraphPatternNotTriples. This is done in remaining sections of this specification.**

## 3.16 Changes to 4.1, Group graph patterns

1. DELETE THE DEFINITION OF GROUP GRAPH PATTERN:

~~Definition: Group Graph Pattern~~

~~A group graph pattern GP is a set of graph patterns, GPi.~~

~~A solution of Group Graph Pattern GP on graph G is any solution S such that, for every element GPi of GP, S is a solution of GPi.~~

*[NOTE to the proposal reader: covered above.]*

2. EDIT THE NEXT PARAGRAPH AS FOLLOWS:

**A group graph pattern is an instance of [19] GroupGraphPattern:**

```
GroupGraphPattern ::= '{' GraphPattern '}'
```

**Semantically, the solutions, and the cardinality of each solution, of the GroupGraphPattern are the same as the solutions and cardinalities of solutions of the GraphPattern.**

For any solution, the same variable is given the same value everywhere in the set of graph patterns making up the group graph pattern. For example, this query has a group graph pattern of one basic graph pattern as the query pattern.

In a SPARQL query string, a group graph pattern is delimited with braces: { }.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  {
          ?x foaf:name ?name .
          ?x foaf:mbox ?mbox .
       }
```

The same solutions would be obtained from a query that grouped the triple patterns in basic graph patterns as below:

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { { ?x foaf:name ?name . }
         { ?x foaf:mbox ?mbox . }
       }
```

3. APPEND THE FOLLOWING:

**However, blank node identifiers have scope confined to a single BlockOfTriples, and do not match across multiple BlockOfTriples. Therefore braces cannot be erased from graph patterns that involve blank node identifiers without a change in semantics. For example**

```
{ ?x <v> _:a . ?y <v> _:a . }
```

**is not in general equivalent to**

```
{ { ?x <v> _:a . } { ?y <v> _:a . } }
```

**The difference can be seen with the following datset:**

```
<x> <v> 1 . <y> <v> 2 .
```

**The query with a single BlockOfTriples has no solutions on this dataset, since there is no way to map _:a to a single node and solve both triple patterns, whereas the query with two BlockOfTriples has one solution, mapping ?x to <x> and ?y to <y>.**

## 3.17 Changes to 5.3, Multiple optional graph patterns

1.

*[NOTE to the proposal reader: This section plainly needs some work to clarify the meaning of multiple OPTIONAL and nested OPTIONAL. Since my proposed semantics below is only a draft, I do not suggest any concrete changes here at this time.*

## 3.18 Changes to 5.4, Optional matching - formal definition

1. REPLACE THE FORMAL DEFINITION WITH THE FOLLOWING:

**Given a graph pattern P of the form**

```
FilteredBasicGraphPattern
OPTIONAL GroupGraphPattern
```

**let FBGP be the FilteredBasicGraphPattern and let GGP be the GroupGraphPattern. Then S is a solution of P if the following two conditions hold:**

**1) S is a solution of FBGP.**

**2) Either of the following is true:**

**a) S is a solution of GGP.**

**b) There is no mapping T that is a solution of both FBGP and GGP, and S is undefined on all variables that appear in GGP but not in FBGP.**

**The cardinality of S is the cardinality of S as a solutin of GGP, if condition 2)a) holds, or 1 if condition 2)b) holds. Note that the cardinality of S can be thought of as the product of its cardinality as a solution to FBGP times its cardinality as a solution to GGP, regarding the latter as 1 if condition 2)b) holds.**

**As usual, note that blank node identifiers are scoped to BlockOfTriples. In particular, using the binding technique for simple entailment, it is not necessary that a blank node identifier be bound to the same RDF term in FilteredBasicGraphPattern and in GroupGraphPattern.**

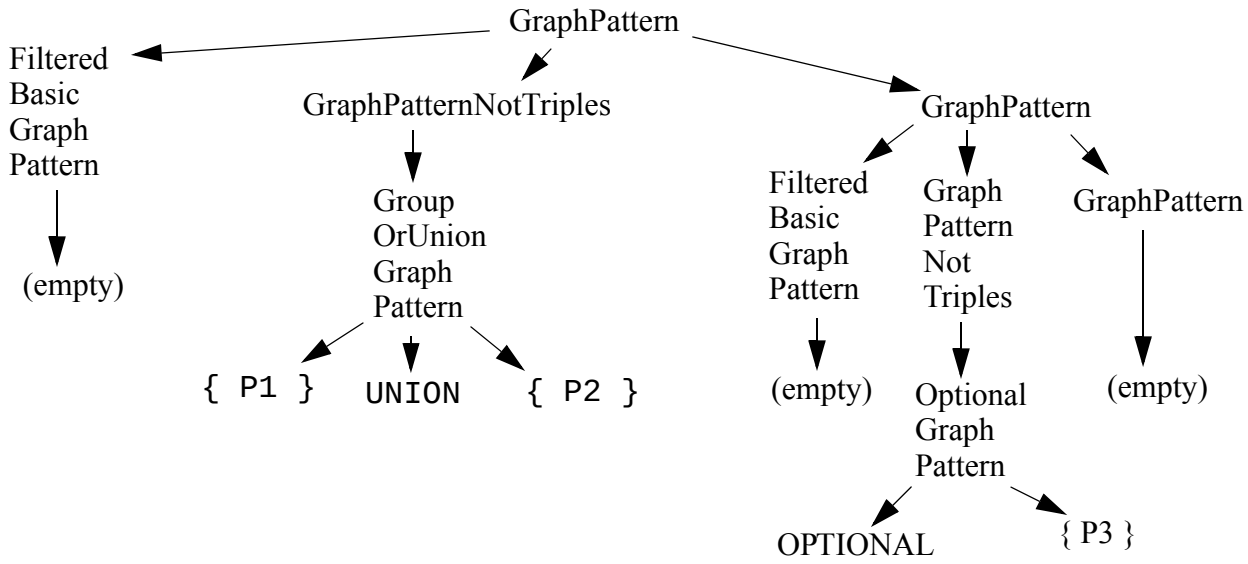
**Discussion:** note that I do not define the semantics of a pattern P of the form

```
GraphPattern OPTIONAL GroupGraphPattern
```

This is because I am endeavoring to make the semantics correspond to the parsing of the syntax. For example, consider

```
{ P1 } UNION { P2 } OPTIONAL { P3 }
```

where P1, P2 and P3 are each a BlockOfTriples. The example is parsable, and is parsed as shown below:




Looking at this diagram, we see that the first operand of the OPTIONAL is the empty FilteredBasicGraphPattern.

## 3.19 Changes to 6.2, UNION matching - formal definition

1. REPLACE THIS SECTION WITH THE FOLLOWING

**Given a graph pattern P of the form**

```
GroupGraphPattern UNION GroupGraphPattern
```

**let GGP1 and GGP2 be the first and second GroupGraphPatterns. Then S is a solution of P is S is a solution of GGP1 or a solution of GGP2. The cardinality of S is computed as follows:**

**1) If S is not a solution of GGP1, let N1 be 0; otherwise let N1 be the cardinality of S as a solution of GGP1.**

**1) If S is not a solution of GGP2, let N2 be 0; otherwise let N2 be the cardinality of S as a solution of GGP1.**

**The cardinality of S is N1 + N2.**

## 3.20 Changes to 8, Querying the dataset

1. REPLACE THE DEFINITION OF RDF DATASET GRAPH PATTERN WITH THE FOLLOWING

**Definition: RDF dataset graph pattern**

**An RDF dataset graph pattern is a character string conforming to the BNF for [25] GraphGraphPattern.**

*[NOTE to the proposal reader: perhaps we should change the name of the BNF nonterminal to DatasetGraphPattern?]*

**If DGP is a GraphGraphPattern, let V be the Var~~OrBlankNode~~OrIRIref contained in DGP and let GGP be the group graph pattern contained in DGP. Let D = { G, (<u1>, G1), ... } be the dataset. S is a solution of DGP with cardinality N if either of the following is true:**

**1) V is a variable, S is defined on V, S(V) = <ui> for some i, and S is a solution of GGP on dataset D' = { Gi, (<u1>, G1), ... } with cardinality N.**

**2) V is a URI = <ui> for some i, and S is a solution of GGP on dataset D' = { Gi, (<u1>, G1), ... } with cardinality N.**

*[NOTE to the proposal reader: note that I am deleting the possibility of using a blank node identifier to identify the graph. The scope of a blank node identifier is a basic graph pattern. Since an RDF dataset graph pattern cannot occur within a basic graph pattern, the semantics appear to be undefinable.]*

## 3.21 Changes to 9, Specifying RDF datasets

1. REARRANGE THE DOCUMENT TO PLACE THIS SECTION BEFORE SECTION 8.

*[NOTE to the proposal reader: logically, you have to know how to specify a dataset before you can talk about how to query it.]*

## 3.22 Changes to 9.1, Specifying the default graph

1. APPEND THE FOLLOWING

**If there is no FROM clause, then the default graph is implementation-defined, and there are no named graphs.**

*[NOTE to the proposal reader: not specified currently, to my knowledge.]*

## 3.23 Changes to 9.2, Specifying named graphs

1. APPEND THE FOLLOWING:

**It is permitted to have redundant named graphs, as in this example:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?src ?name

FROM NAMED <http://example.org/alice>
FROM NAMED <http://example.org/alice>

WHERE
{ GRAPH ?src { ?x foaf:name ?name } }
```

**In the preceding example, the graph <http://example.org/alice> has been named twice. This is equivalent to naming it once.**

*[NOTE to the proposal reader: not specified currently whether this is an error or not.]*

## 3.24 Changes to 9.3, Combining FROM and FROM NAMED

1. APPEND THE FOLLOWING

**The formal semantics of FROM and FROM NAMED is as follows:**

**If a SPARQL query contains no DataSetClause, then its dataset is D = { G }, where G is an implementation-defined graph. There are no named graphs in this case.**

**If a SPARQL query contains one or more DataSetClauses, then its dataset is D = {G, (<u1>, G1), ... } where:**

**1) G is the RDF merge of all graphs named in any DefaultGraphClause, or the implementation-defined default graph if there is no DefaultGraphClause, and**

**2) for each NamedGraphClause, there is an i such that <ui> is SourceSelector of that NamedGraphClause, and Gi is the graph identified by <ui>.**

## 3.25 Changes to A.7, Grammar

1. IN THE BNF FOR GRAPHGRAPHPATTERN, CHANGE VARORBLANKNODEORIRIREF TO VARORIRIREF:

```
GraphGraphPattern ::=
     'GRAPH' ~~VarOrBlankNodeOrIRIref~~ VarOrIRIref
     GroupGraphPattern
```

*[NOTE to the proposal reader: the semantics of GraphGraphPattern is not defined if a blank node identifier is used to select the graph. However, a blank node identifier is scoped to a BasicGraphPattern, so there is no way to assign a meaning to a blank node identifier in this context.]*

2. I HAVE NOT CHECKED IF IT IS NOW POSSIBLE TO DELETE VARORBLANKNODEORIRIREF.

***- End of paper -***