```
define([
    "./graphite/api"
], function (API) {
    /*!
     * Graphite Core
     * Copyright (C) 2012 Arne Hassel
     * MIT Licensed
     *
     * The core Graphite module.
     * Based on the graphite-library, http://graphitejs.com/, designed by Alex Young
     *
     * The graphite object.
     *
     * @returns {Object} The API module
     */
    return function Constructor(input) {
        return new API(input);
    };
});define([
    "./dictionary",
    "./graph",
    "./query",
    "./utils",
    "./when"
], function (Dictionary, Graph, Query, Utils, When) {
    var Api = function () {
            return new Api.prototype.init();
        },
        ApiSubject = function (api, subjectName) {
            return new ApiSubject.prototype.init(api, subjectName);
        };
    Api.prototype = {
        init: function () {
            this.g = Graph();
            this.q = Query();
        },
        /**
         *
         * @param subject
         * @param predicate
         * @param object
         * @param [callback]
         * @return {*}
         */
        addStatement: function (subject, predicate, object, callback) {
            this.q = Query("INSERT DATA {{0}}".format(Dictionary.createStatement({
                    subject: subject,
                    predicate: predicate,
                    object: object
                })));
            return this.execute(callback, function () {
                this.q = Query();
            }.bind(this));
        },
        base: function (uri) {
            //console.log("IN API, BASE BEGINNING");
            this.q.base.apply(this.q, arguments);
            return this;
        },
        execute: function (callback, onsuccess) {
            this.g.execute(this.q, callback, onsuccess);
            return this;
        },
        filter: function (filter) {
            this.q.filter.apply(this.q, arguments);
            return this;
        },
```

```
        getSubject: function (subjectName) {
            return new ApiSubject(this, subjectName);
        },
        group: function (group) {
            this.q.group.apply(this.q, arguments);
            return this;
        },
        listStatements: function (options, callback) {
            if (!callback && Utils.isFunction (options)) {
                callback = options;
                options = {};
            }
            var subject = options.subject ? Dictionary.createSubject(options.subject).toNT() : "?subject",
                predicate = options.predicate ? Dictionary.createPredicate(options.predicate).toNT() : "?
predicate",
                object = options.object ? Dictionary.createObject(options.object).toNT() : "?object";
            this.q = Query("SELECT * WHERE { {0} {1} {2} }".format(subject, predicate, object));
            return this.execute(callback);
        },
        load: function (uri, callback) {
            this.q = Query("LOAD <" + uri + ">");
            return this.execute(callback);
        },
        optional: function (optional) {
            this.q.optional.apply(this.q, arguments);
            return this;
        },
        prefix: function (prefix, local) {
            this.q.prefix.apply(this.q, arguments);
            return this;
        },
        query: function (queryString) {
            this.q = Query.apply(this.q, arguments);
            return this;
        },
        /**
         *
         * @param variable
         * @param pattern
         * @param [flags]
         */
        regex: function (variable, pattern, flags) {
            this.q.regex(variable, pattern, flags);
            return this;
        },
        removeStatement: function (subject, predicate, object) {
            this.q = Query("DELETE DATA { {0} }".format(Dictionary.createStatement({
                subject: subject,
                predicate: predicate,
                object: object
            })));
            return this.execute();
        },
        select: function (projection) {
            this.q.select(projection);
            return this;
        },
        size: function (callback) {
            this.g.size(callback);
            return this;
        },
        then: function (callback) {
            this.g.then(function () {
                callback(this);
            }.bind(this));
            return this;
        },
```

```
        where: function (pattern) {
            this.q.where(pattern);
            return this;
        }
    };
    Api.prototype.init.prototype = Api.prototype;
    ApiSubject.prototype = {
        init: function (api, subjectName) {
            this.api = api;
            this.query = this.api.q.getSubject(subjectName);
        },
        execute: function (callback) {
            this.api.execute(this.query, callback);
        },
        then: function (callback) {
            this.api.then(callback);
        }
    };
    ApiSubject.prototype.init.prototype = ApiSubject.prototype;
    return Api;
});define([
    "./../rdfquery/uri",
    "./utils"
], function (Uri, Utils) {
    function getDataType(value) {
        if(Utils.isBoolean(value)) {
            return Dictionary.Symbol.prototype.XSDboolean;
        } else if (Utils.isInteger(value)) {
            //console.log("IN DICTIONARY, DATATYPE IS INTEGER", value);
            return Dictionary.Symbol.prototype.XSDinteger;
        } else if (Utils.isDouble(value)) {
            //console.log("IN DICTIONARY, DATATYPE IS DOUBLE", value);
            return Dictionary.Symbol.prototype.XSDfloat;
        }
        return null;
    }
    var Dictionary = {
        /**
         *
         * @param [object]
         * @param [options]
         * @return {*}
         */
        createObject: function (object, options) {
            options = options || {};
            if (object && !Utils.isString(object)) {
                if (options.base) {
                    return this.Symbol(Uri('' + object, options.base));
                }
                if (options["isBlankNode"]) {
                    return this.BlankNode(object);
                }
                //console.log("IN DICTIONARY, OBJECT FALLS TO LITERAL");
                return this.Literal(object);
            }
            if (object && options.base) {
                return this.Symbol(Uri(object, options.base));
            }
            if (object && Utils.isUri(object)) {
                return this.Symbol(object);
            }
            if (object) {
                //console.log("IN DICTIONARY, OBJECT IS LITERAL", object);
                return this.Literal(object);
            }
            return this.BlankNode();
        },
```

```
    /**
     *
     * @param predicate
     * @param [base]
     * @return {*}
     */
    createPredicate: function (predicate, base) {
        return this.Symbol(Uri(predicate, base));
    },
    /**
     *
     * @param parts
     * @return {*}
     */
    createStatement: function (parts) {
        var subject = this.createSubject(parts.subject),
            predicate = this.createPredicate(parts.predicate),
            object = this.createObject(parts.object);
        return Dictionary.Statement(subject, predicate, object);
    },
    /**
     *
     * @param [subject]
     * @param [base]
     * @return {*}
     */
    createSubject: function (subject, base) {
        if (subject) {
            if (!Utils.isString(subject)) {
                return this.BlankNode(subject);
            }
            return this.Symbol(Uri(subject, base).toString());
        }
        return this.BlankNode();
    },
    //      Convert Javascript representation to RDF term object
    //
    createTerm: function (val, graph) {
        if (typeof val == 'object') {
            if (val instanceof Date) {
                var d2=function (x) {
                    return(''+(100+x)).slice(1,3);
                };  // format as just two digits
                return Dictionary.Literal(
                    ''+ val.getUTCFullYear() + '-'+
                        d2(val.getUTCMonth()+1) +'-'+d2(val.getUTCDate())+
                        'T'+d2(val.getUTCHours())+':'+d2(val.getUTCMinutes())+
                        ':'+d2(val.getUTCSeconds())+'Z',
                    undefined, Dictionary.Symbol.prototype.XSDdateTime);
            } else if (val instanceof Array) {
                var x = Dictionary.Collection(graph);
                for (var i=0; i<val.length; i++) {
                    x.append(Dictionary.createTerm(val[i], graph));
                }
                return x;
            } else {
                return val;
            }
        }
        if (typeof val == 'string') {
            return Dictionary.Literal(val);
        }
        if (typeof val == 'number') {
            //console.log("IN DICTIONARY, NUMBER");
            var dt;
            if (((''+val).indexOf('e')>=0) {
                dt = Dictionary.Symbol.prototype.XSDfloat;
```

```
                } else if (((''+val).indexOf('.')>=0) {
                    dt = Dictionary.Symbol.prototype.XSDdecimal;
                } else {
                    dt = Dictionary.Symbol.prototype.XSDinteger;
                }
                return Dictionary.Literal(val, undefined, dt);
            }
            if (typeof val == 'boolean') {
                return Dictionary.Literal(val ? "1": "0", undefined,
Dictionary.Symbol.prototype.XSDboolean);
            }
            if (typeof val == 'undefined') {
                return undefined;
            }
            throw ("Can't make term from " + val + " of type " + typeof val);
        }
    };
    //  Blank Node
    if (typeof Dictionary.NextId != 'undefined') {
        Dictionary.log.error('Attempt to re-zero existing blank node id counter at '+Dictionary.NextId);
    } else {
        Dictionary.NextId = 0;  // Global genid
    }
    Dictionary.NTAnonymousNodePrefix = "_:";
    Dictionary.BlankNode = function (id) {
        return new Dictionary.BlankNode.prototype.init(id);
    };
    Dictionary.BlankNode.prototype.init = function ( id ) {
        this.id = Dictionary.NextId++;
        this.value = id ? id : this.id.toString();
        return this
    };
    //Dictionary.BlankNode.prototype.termType = 'bnode';
    Dictionary.BlankNode.prototype.toNT = function () {
        return Dictionary.NTAnonymousNodePrefix + this.id
    };
    Dictionary.BlankNode.prototype.toString = Dictionary.BlankNode.prototype.toNT;
    Dictionary.BlankNode.prototype.toQuads = function () {
        return {'blank': Dictionary.NTAnonymousNodePrefix + this.id};
    };
    Dictionary.BlankNode.prototype.init.prototype = Dictionary.BlankNode.prototype;
    // Collection
    Dictionary.Collection = function (graph) {
        return new Dictionary.Collection.prototype.init(graph);
    };
    Dictionary.Collection.prototype.init = function (graph) {
        this.id = Dictionary.NextId++;  // Why need an id? For hashstring.
        this.elements = [];
        this.closed = false;
        this.graph = graph;
    };
    Dictionary.Collection.idCounter = 0;
    //Dictionary.Collection.prototype.termType = 'collection';
    Dictionary.Collection.prototype.toNT = function () {
        return Dictionary.NTAnonymousNodePrefix + this.id
    };
    Dictionary.Collection.prototype.toQuads = function () {
        var acum = [];
        var subjectId = "_:list"+Dictionary.Collection.idCounter;
        Dictionary.Collection.idCounter++;
        var first = {'uri': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#first'};
        var rest = {'uri': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#rest'};
        var nil = {'uri': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#nil'};
        var subject;
        var nextSubject = function (i) {
            return {
                'blank': subjectId+"p"+i
```

```
            }
        };
        for (var i=0; i<this.elements.length; i++) {
            subject = nextSubject(i);
            if(i<(this.elements.length-1)) {
                nextSubject = nextSubject(i + 1);
            } else {
                nextSubject = nil;
            }
            acum.push({
                'subject': subject,
                'predicate': first,
                'object': this.elements[i].toQuads(),
                'graph': this.graph
            });
            acum.push({
                'subject': subject,
                'predicate': rest,
                'object': nextSubject,
                'graph': this.graph
            });
        }
        return acum;
    };
    Dictionary.Collection.prototype.append = function (el) {
        this.elements.push(el)
    };
    Dictionary.Collection.prototype.unshift=function (el){
        this.elements.unshift(el);
    };
    Dictionary.Collection.prototype.shift=function (){
        return this.elements.shift();
    };
    Dictionary.Collection.prototype.close = function () {
        this.closed = true
    };
    Dictionary.Collection.prototype.init.prototype = Dictionary.Collection.prototype;
    // These are the classes corresponding to the RDF and N3 data models
    //
    // Designed to look like rdflib and cwm designs.
    //
    // Issues: Should the names start with RDF to make them
    //      unique as program-wide symbols?
    //
    // W3C open source licence 2005.
    //
    //  Symbol
    Dictionary.Empty = function () {
        return new Dictionary.Empty.prototype.init();
    };
    Dictionary.Empty.prototype.init = function () {
        return this;
    };
    //Dictionary.Empty.prototype.termType = 'empty';
    Dictionary.Empty.prototype.toNT = function () { return "()" };
    Dictionary.Empty.prototype.toQuads = function () {
        return {
            'uri': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#nil'
        };
    };
    Dictionary.Empty.prototype.init.prototype = Dictionary.Empty.prototype;
    //  Formula
    //
    //  Set of statements.
    Dictionary.Formula = function (graph) {
        return new Dictionary.Formula.prototype.init(graph);
    };
```

```
Dictionary.Formula.prototype.init = function (graph) {
    this.statements = [];
    this.graph = graph;
    return this;
};
//Dictionary.Formula.prototype.termType = 'formula';
Dictionary.Formula.prototype.toNT = function () {
    var statements = Utils.map(this.statements, function (s) {
        return s.toNT();
    });
    return "{\n" + statements.join('\n') + "\n}";
};
Dictionary.Formula.prototype.toQuads = function () {
    var accumulated = [];
    for(var i=0; i<this.statements.length; i++) {
        var nextValue = this.statements[i].toQuads();
        if(nextValue.constructor === Array) {
            accumulated = accumulated.concat(nextValue);
        } else {
            accumulated.push(nextValue);
        }
    }
    return accumulated;
};
Dictionary.Formula.prototype.add = function (subj, pred, obj, why) {
    this.statements.push(Dictionary.Statement(subj, pred, obj, why, this.graph));
};
// Convenience methods on a formula allow the creation of new RDF terms:
Dictionary.Formula.prototype.sym = function (uri) {
    return Dictionary.Symbol(uri)
};
Dictionary.Formula.prototype.literal = function (val, lang, dt) {
    if(dt != null && dt.value != null && dt.value.indexOf("http://") === -1) {
        for(var ns in this.namespaces) {
            if(dt.value.indexOf(ns) === 0 && this.namespaces.hasOwnProperty(ns)) {
                dt.value = this.namespaces[ns]+(dt.value.split(ns+":")[1]);
                break;
            }
        }
    }
    //console.log("IN DICTIONARY, FORMULA LITERAL DATATYPE", dt);
    return Dictionary.Literal(''+val, lang, dt)
};
Dictionary.Formula.prototype.bnode = function (id) {
    return Dictionary.BlankNode(id)
};
Dictionary.Formula.prototype.formula = function () {
    return Dictionary.Formula(this.graph);
};
Dictionary.Formula.prototype.collection = function () { // obsolete
    return Dictionary.Collection(this.graph)
};
Dictionary.Formula.prototype.list = function (values) {
    var li = Dictionary.Collection(this.graph);
    if (values) {
        for(var i = 0; i<values.length; i++) {
            li.append(values[i]);
        }
    }
    return li;
};
Dictionary.Formula.prototype.init.prototype = Dictionary.Formula.prototype;
//  Literal
Dictionary.Literal = function (value, lang, datatype) {
    //console.log("IN DICTIONARY, LITERAL INIT", value, lang, datatype);
    return new Dictionary.Literal.prototype.init(value, lang, datatype);
};
```

```
Dictionary.Literal.prototype.init = function (value, lang, datatype) {
    this.value = value;
    this.lang = lang;
    //console.log("IN DICTIONARY, LITERAL DATATYPE BEFORE", datatype);
    this.datatype = datatype || getDataType(value);
    //console.log("IN DICTIONARY, LITERAL DATATYPE AFTER", this.datatype);
    return this;
};
//Dictionary.Literal.prototype.termType = 'literal';
Dictionary.Literal.prototype.toNT = function () {
    var str = this.value;
    if (typeof str != 'string') {
        if (typeof str == 'number') return ''+str;
        throw Error("Value of RDF literal is not string: "+str);
    }
    str = str.replace(/\\/g, '\\\\');  // escape backslashes
    str = str.replace(/"/g, '\\"');    // escape quotes
    str = str.replace(/\n/g, '\\n');    // escape newlines
    str = '"' + str + '"';  //';
    if (this.datatype){
        //console.log("DATATYPE", this.datatype, this.datatype.toNT);
        str = str + '^^{0}'.format(this.datatype.toNT());
    } else if (this.lang) {
        str = str + "@{0}".format(this.lang);
    }
    return str;
};
Dictionary.Literal.prototype.toQuads = function () {
    var str = this.value;
    if (typeof str != 'string') {
        if (typeof str == 'number') {
            return ''+str;
        }
        throw Error("Value of RDF literal is not string: "+str);
    }
    str = str.replace(/\\/g, '\\\\');  // escape backslashes
    str = str.replace(/"/g, '\\"');    // escape quotes
    str = str.replace(/\n/g, '\\n');    // escape newlines
    str = '"' + str + '"';  //';
    if (this.datatype){
        str = str + '^^<{0}>'.format(this.datatype.value);
    } else if (this.lang) {
        str = str + '@{0}'.format(this.lang);
    }
    return {
        'literal': str
    };
};
Dictionary.Literal.prototype.init.prototype = Dictionary.Literal.prototype;
//  Statement
//
//  This is a triple with an optional reason.
//
//   The reason can point to provenece or inference
//
Dictionary.Statement = function (subject, predicate, object, why, graph) {
    return new Dictionary.Statement.prototype.init(subject, predicate, object, why, graph);
};
Dictionary.Statement.prototype.init = function (subject, predicate, object, why, graph) {
    this.subject = Dictionary.createTerm(subject, graph);
    this.predicate = Dictionary.createTerm(predicate, graph);
    this.object = Dictionary.createTerm(object, graph);
    if (typeof why !='undefined') {
        this.why = why;
    }
    this.graph = graph;
    return this;
```

```
    };
    Dictionary.Statement.prototype.toString = Dictionary.Statement.prototype.toNT = function () {
        //console.log("SUBJECT", this.subject, this.subject.toNT);
        //console.log("PREDICATE", this.predicate, this.predicate.toNT);
        //console.log("OBJECT", this.object, this.object.toNT);
        return (this.subject.toNT() + " "
            + this.predicate.toNT() + " "
            +  this.object.toNT() +" .");
    };
    Dictionary.Statement.prototype.toQuads = function () {
        var object = this.object.toQuads();
        if(object.constructor === Array) {
            var nextObject = object[0].subject;
            object.push({
                'subject': this.subject.toQuads(),
                'predicate': this.predicate.toQuads(),
                'object': nextObject,
                'graph': this.graph
            });
            return object;
        } else {
            return {
                'subject': this.subject.toQuads(),
                'predicate': this.predicate.toQuads(),
                'object': this.object.toQuads(),
                'graph': this.graph
            };
        }
    };
    Dictionary.Statement.prototype.init.prototype = Dictionary.Statement.prototype;
    // Symbol
    Dictionary.Symbol = function (uri) {
        return new Dictionary.Symbol.prototype.init(uri);
    };
    Dictionary.Symbol.prototype.init = function ( uri ) {
        this.uri = uri;
        this.value = uri;   // -- why? -tim
        return this;
    };
    Dictionary.Symbol.prototype.init.prototype = Dictionary.Symbol.prototype;
    //Dictionary.Symbol.prototype.termType = 'symbol';
    Dictionary.Symbol.prototype.toNT = function () {
        return ("<" + this.uri + ">");
    };
    Dictionary.Symbol.prototype.toQuads = function () {
        return {
            token:'uri',
            prefix:null,
            suffix:null,
            value:this.uri
        };
    };
    //  Some precalculated symbols
    Dictionary.Symbol.prototype.XSDboolean = Dictionary.Symbol('http://www.w3.org/2001/
XMLSchema#boolean');
    Dictionary.Symbol.prototype.XSDdecimal = Dictionary.Symbol('http://www.w3.org/2001/
XMLSchema#decimal');
    Dictionary.Symbol.prototype.XSDfloat = Dictionary.Symbol('http://www.w3.org/2001/XMLSchema#float');
    Dictionary.Symbol.prototype.XSDinteger = Dictionary.Symbol('http://www.w3.org/2001/
XMLSchema#integer');
    Dictionary.Symbol.prototype.XSDdateTime = Dictionary.Symbol('http://www.w3.org/2001/
XMLSchema#dateTime');
    Dictionary.Symbol.prototype.integer = Dictionary.Symbol('http://www.w3.org/2001/
XMLSchema#integer'); // Used?
    Dictionary.Symbol.prototype.init.prototype = Dictionary.Symbol.prototype;
    return Dictionary;
});define([
```

```
    "./dictionary",
    "./../rdfstore/rdf-persistence/lexicon",
    "./../rdfstore/rdf-persistence/quad_backend",
    "./../rdfstore/query-engine/query_engine",
    "./serializer/sparql",
    "./utils",
    "./when"
], function (Dictionary, Lexicon, QuadBackend, QueryEngine, Serializer, Utils, When) {
    "use strict";
    function bindVar (vars) {
        return Utils.map(vars, function (v) {
            if (v && v.hasOwnProperty("value")) {
                return v.value;
            }
            return null;
        });
    }
    function getExecuteFunction(deferred, graph, query, callback, onsuccess) {
        var graph1,
            graph2,
            queryKind = getQueryKind(query);
        switch(queryKind) {
            case "ask":
                return function (success, result) {
                    graph1 = graph.clone();
                    callback(result);
                    deferred.resolve(graph1);
                };
            case "construct":
                return function (success, results) {
                    graph1 = graph.clone();
                    graph2 = Graph(results.triples);
                    callback(graph2);
                    deferred.resolve(graph1);
                };
            case "deletedata":
                return function () {
                    Graph().execute(query, function (g) {
                        callback(g);
                    });
                    graph1 = graph.clone();
                    deferred.resolve(graph1);
                };
            case "insertdata":
                return function () {
                    if(callback) {
                        graph2 = Graph().execute(query);
                        callback(graph2);
                    }
                    graph1 = graph.clone();
                    deferred.resolve(graph1);
                };
            case "load":
                return function (success, results) {
                    //console.log("IN GRAPH, getExecuteFunction", results);
                    graph1 = graph.clone();
                    deferred.resolve(graph1);
                    /*
                    graph1 = graph.clone().load(results);
                    deferred.resolve(graph1);
                    if (callback) {
                        graph2 = Graph(results);
                        callback(graph2);
                    }
                    */
                };
            case "select":
```

```
            return function (success, results) {
                var vars, lvars;
                if (callback && callback.length > 0) {
                    vars = Utils.extractArgumentMap(callback);
                    Utils.each(results, function (args) {
                        try {
                            lvars = Utils.mapArgs(vars, args);
                        } catch (e) {
                            lvars = args;
                        }
                        lvars = bindVar(lvars);
                        callback.apply(graph, lvars);
                    });
                } else if (callback) {
                    Utils.each(results, callback);
                }
                if (onsuccess) {
                    onsuccess(graph);
                }
                deferred.resolve(graph);
            };
        default:
            //console.log("Query not supported", queryKind);
            throw new Error("Query not supported: " + queryKind);
            deferred.resolve(graph);
    }
}
function getTriples (callback) {
    var formula = Dictionary.Formula(),
        subject,
        predicate,
        object;
    this.execute("SELECT * WHERE { ?s ?p ?o }", function (s, p, o) {
        subject = Dictionary.createSubject(s);
        predicate = Dictionary.createPredicate(p);
        object = Dictionary.createObject(o);
        formula.add(subject, predicate, object);
    }, function () {
        callback(formula);
    });
    return this;
}
function getQueryKind(query) {
    if(!Utils.isString(query)) {
        return query.units[0].kind;
    }
    var kind = null,
        currentPos = query.length,
        position = {
            ask: query.indexOf("ASK"),
            construct: query.indexOf("CONSTRUCT"),
            insertdata: query.indexOf("INSERT DATA"),
            load: query.indexOf("LOAD"),
            select: query.indexOf("SELECT")
        };
    Utils.each(position, function (pos, type) {
        kind = pos !== -1 && pos < currentPos ?
            type :
            kind;
    });
    return kind;
}
function loadFormula(graph, formula, deferred) {
    //console.log("IN GRAPH, loadFormula", formula.toNT());
    graph.engine.execute('INSERT DATA ' + formula.toNT(), function () {
        deferred.resolve(graph);
    });
```

```
    }
    function loadTriples(graph, triples, deferred) {
        //console.log("IN GRAPH, loadTriples", triples);
        graph.engine.execute('INSERT DATA {' + triples.join("\n") + '}', function () {
            deferred.resolve(graph);
        });
    }
    function loadUri(graph, uri, deferred) {
        //console.log("IN GRAPH, loadUri", uri);
        graph.engine.execute("LOAD <" + uri + ">", function (success, results) {
            loadTriples(graph, results.statements, deferred);
        });
    }
    function loadUris(graph, uris, deferred) {
        var promises = Utils.map(uris, function () {
            return When.defer();
        });
        Utils.each(uris, function (uri, i) {
            loadUri(graph, uri, promises[i]);
        });
        When.all(promises).then(function () {
            deferred.resolve(graph);
        });
    }
    /*
     * The graph object
     */
    var Graph = function (input) {
        return new Graph.prototype.init(input);
    };
    Graph.prototype = {
        init: function (input) {
            var self = this;
            this.deferred = When.defer();
            new Lexicon.Lexicon(function(lexicon){
                self.lexicon = lexicon;
                new QuadBackend.QuadBackend({
                    treeOrder: 2
                }, function(backend){
                    self.engine = new QueryEngine.QueryEngine({
                        backend: backend,
                        lexicon: lexicon
                    });
                    self.deferred.resolve(self);
                    if (input && input instanceof Graph) {
                        getTriples.call(input, function (formula) {
                            self.load(formula);
                        });
                    } else if (input) {
                        self.load(input);
                    }
                });
            });
            return this;
        },
        /**
         * Clones an instance of a graph, returning a new instance
         * @return {*}
         */
        clone: function () {
            return Graph(this);
        },
        /**
         * The execute method takes a query and a callback. It returns a promise
         * that resolves to a graph. The constructed graph will reflect an
         * altered graph or a scoped graph, depending on the type of query, as
         * described below:
```

```
         *
         * <ul>
         *     <li>ASK: returns the graph queried on, the callback will return a
         *     boolean</li>
         *     <li>CLEAR: returns the altered graph (an empty graph). Callback
         *     is the graph itself.</li>
         *     <li>CONSTRUCT: returns and callbacks the constructed graph.</li>
         *     <li>DELETE DATA: returns the altered graph. Callback is a graph
         *     with the deleted triples.</li>
         *     <li>DESCRIBE: returns and callbacks the constructed graph.</li>
         *     <li>INSERT DATA: returns the altered graph. Callback is a graph
         *     with the inserted triples.</li>
         *     <li>LOAD: returns the altered graph. Callback is the loaded graph.</li>
         *     <li>SELECT: returns graph queried on. The callback is the selected
         *     variables.</li>
         * </ul>
         *
         * @param query
         * @param [callback]
         * @param [onsuccess]
         * @return {*}
         */
        execute: function (query, callback, onsuccess) {
            var deferred = When.defer();
            query = query.retrieveTree ? query.retrieveTree() : query;
            this.deferred.then(function (graph) {
                graph.engine.execute(query, getExecuteFunction(deferred, graph, query, callback,
onsuccess));
            });
            this.deferred = deferred;
            return this;
        },
        /**
         *
         * @param {String} input
         */
        load: function (input) {
            var deferred = When.defer();
            this.deferred.then(function (graph) {
                if (input instanceof Graph) {
                    getTriples.call(input, function (formula) {
                        loadFormula(graph, formula, deferred);
                    });
                } else if (Utils.isString(input) && Utils.isUri(input)) {
                    loadUri(graph, input, deferred);
                } else if (Utils.isFunction(input.toNT)) {
                    loadFormula(graph, input, deferred);
                } else if (Utils.isArray(input) && input.length > 0) {
                    if (Utils.isString(input[0])) {
                        loadUris(graph, input, deferred);
                    } else {
                        loadTriples(graph, input, deferred);
                    }
                } else {
                    throw new Error ("Input not supported: " + input);
                }
            });
            this.deferred = deferred;
            return this;
        },
        merge: function (graphToMerge) {

            this.deferred.then(function (graph) {
                getTriples.call(graphToMerge, function (formula) {
                    graph.load(formula);
                });
            });
```

```
            return this;
        },
        /**
         *
         * @param {Function} callback
         * @return {*}
         */
        size: function (callback) {
            this.deferred.then(function (graph) {
                graph.engine.execute("SELECT * WHERE { ?s ?p ?o }", function (success, results) {
                    callback(results.length);
                });
            });
            return this;
        },
        /**
         *
         * @param {Function} callback
         * @return {*}
         */
        then: function (callback) {
            this.deferred.then(function (graph) {
                callback(graph);
            });
            return this;
        }
    };
    Graph.prototype.init.prototype = Graph.prototype;
    return Graph;
});/*global define */
define([
    "./dictionary",
    "./../rdfstore/rdf-persistence/lexicon",
    "./../rdfstore/rdf-persistence/quad_backend",
    "./../rdfstore/query-engine/query_engine",
    "./utils",
    "./when"
], function (Dictionary, Lexicon, QuadBackend, QueryEngine, Utils, When) {
    "use strict";
    function bindVar (vars) {
        return Utils.map(vars, function (v) {
            if (v && v.hasOwnProperty("value")) {
                return v.value;
            }
            return null;
        });
    }
    function executeAsk (promise, options) {
        return function (success, result) {
            promise.resolve(options.graph);
            if (options.callback) {
                options.callback(result);
            }
        };
    }
    function executeConstruct (promise, options) {
        return function (success, results) {
            promise.resolve(options.graph);
            if(options.callback) {
                options.callback(Graph(results.triples));
            }
        }
    }
    function executeDeleteData (promise, options) {
        //console.log("IN GRAPH, DELETE DATA", options.query);
        if (options.callback) {
            Graph().execute(options.query).then(function (g) {
```

```
                options.callback(g);
            });
        }
        return function () {
            //console.log("IN GRAPH, EXECUTE DELETE DATA");
            options.graph.clone().then(function(g) {
                //console.log("IN GRAPH, DELETE CLONED");
                promise.resolve(g);
            });
        };
    }
    function executeInsertData (promise, options) {
        //console.log("IN GRAPH, INSERT DATA", options.query);
        if (options.callback) {
            Graph().execute(options.query).then(function (g) {
                options.callback(g);
            });
        }
        return function () {
            promise.resolve(options.graph);
        };
    }
    function executeLoad (promise, options) {
        var query;
        //console.log("BEFORE LOAD");
        return function (success, results) {
            //console.log("GRAPH LOAD, RESULTS", success, results);
            query = "INSERT DATA " + results.toNT();
            Graph(options.graph).execute(query).then(function (g) {
                if(options.callback) {
                    Graph().then(function (g) {
                        //console.log("ONCALLBACK QUERY", query);
                        g.execute(query, function (g) {
                            options.callback(g);
                        });
                    });
                }
                promise.resolve(g);
            });
        };
    }
    function executeSelect(promise, options) {
        return function (success, results) {
            //console.log("IN GRAPH, SELECT QUERY", results.length, results);
            if (results.length === 0) {
                //console.debug("The query didn't return any results");
            } else {
                //console.debug("IN QUERY, executeSelect", results);
            }
            var vars, lvars;
            if (options.callback && options.callback.length > 0) {
                vars = Utils.extractArgumentMap(options.callback);
                Utils.each(results, function (args) {
                    //console.debug("IN GRAPH, SELECT", args);
                    try {
                        lvars = Utils.mapArgs(vars, args);
                    } catch (e) {
                        lvars = args;
                    }
                    //console.debug("IN GRAPH, SELECT", lvars);
                    options.callback.apply(options.graph, bindVar(lvars));
                });
            } else if (options.callback) {
                Utils.each(results, options.callback);
            }
            if(options.onsuccess) {
                options.onsuccess();
```

```
            }
            promise.resolve(options.graph);
        };
    }
    function getExecuteFunction(query) {
        var queryKind = getQueryKind(query);
        //console.log("IN GRAPH, GET EXECUTE FUNCTION", queryKind);
        switch (queryKind) {
            case "ask": return executeAsk;
            case "construct": return executeConstruct;
            case "deletedata": return executeDeleteData;
            case "insertdata": return executeInsertData;
            case "load": return executeLoad;
            case "select": return executeSelect;
            default: throw new Error("Query not supported!");
        }
    }
    function getTriples (graph) {
        var deferred = When.defer(),
            formula = Dictionary.Formula(),
            subject,
            predicate,
            object;
        graph.engine.execute("SELECT * WHERE { ?s ?p ?o }", function (success, results) {
            Utils.each(results, function (t) {
                subject = Dictionary.createSubject(t.s.value);
                predicate = Dictionary.createPredicate(t.p.value);
                object = Dictionary.createObject(t.o.value);
                formula.add(subject, predicate, object);
            });
            deferred.resolve(formula);
        });
        return deferred;
    }
    function getQueryKind(query) {
        if(!Utils.isString(query)) {
            return query.units[0].kind;
        }
        var kind = null,
            currentPos = query.length,
            position = {
                ask: query.indexOf("ASK"),
                construct: query.indexOf("CONSTRUCT"),
                insertdata: query.indexOf("INSERT DATA"),
                load: query.indexOf("LOAD"),
                select: query.indexOf("SELECT")
            };
        Utils.each(position, function (pos, type) {
            kind = pos !== -1 && pos < currentPos ?
                type :
                kind;
        });
        return kind;
    }
    function loadFormula(graph, resource, deferred) {
        graph.engine.execute('INSERT DATA ' + resource.toNT(), function () {
            deferred.resolve(graph);
        });
    }
    function loadGraph(graph, resource, deferred) {
        getTriples(resource).then(function (formula) {
            graph.engine.execute('INSERT DATA ' + formula.toNT(), function () {
                deferred.resolve(graph);
            });
        });
    }
    function loadStatements(graph, resource, deferred) {
```

```
        graph.engine.execute('INSERT DATA {' + resource.join("\n") + '}', function () {
            deferred.resolve(graph);
        });
    }
    function loadUri(graph, uri, deferred) {
        graph.engine.execute('LOAD <' + uri + '>', function (success, results) {
            loadFormula(graph, results, deferred);
        });
    }
    function loadUris(graph, uris, deferred) {
        if (uris.length > 0) {
            var uri = uris.pop(),
                promise = When.defer();
            loadUri(graph, uri, promise);
            promise.then(function (graph) {
                loadUris(graph, uris, deferred);
            });
        } else {
            deferred.resolve(graph);
        }
    }
    /*
     * The graph object
     */
    var Graph = function (data) {
        return new Graph.prototype.init(data);
    };
    Graph.prototype = {
        init: function (data) {
            this.deferred = When.defer();
            new Lexicon.Lexicon(function(lexicon){
                this.lexicon = lexicon;
                new QuadBackend.QuadBackend({
                    treeOrder: 2
                }, function(backend){
                    this.engine = new QueryEngine.QueryEngine({
                        backend: backend,
                        lexicon: lexicon
                    });
                    this.deferred.resolve(this);
                    if (data) {
                        this.extend(data);
                    }
                }.bind(this));
            }.bind(this));
        },
        clone: function () {
            return Graph(this);
        },
        /**
         * The execute method takes a query and a callback. It returns a promise
         * that resolves to a graph. The constructed graph will reflect an
         * altered graph or a scoped graph, depending on the type of query, as
         * described below:
         *
         * <ul>
         *     <li>ASK: returns the graph queried on, the callback will return a
         *     boolean</li>
         *     <li>CLEAR: returns the altered graph (an empty graph). Callback
         *     is the graph itself.</li>
         *     <li>CONSTRUCT: returns and callbacks the constructed graph.</li>
         *     <li>DELETE DATA: returns the altered graph. Callback is a graph
         *     with the deleted triples.</li>
         *     <li>DESCRIBE: returns and callbacks the constructed graph.</li>
         *     <li>INSERT DATA: returns the altered graph. Callback is a graph
         *     with the inserted triples.</li>
         *     <li>LOAD: returns the altered graph. Callback is the loaded graph.</li>
```

```
         *     <li>SELECT: returns graph queried on. The callback is the selected
         *     variables.</li>
         * </ul>
         *
         * @param query
         * @param [callback]
         * @param [onsuccess]
         * @return {*}
         */
        execute: function (query, callback, onsuccess) {
            query = query.retrieveTree ? query.retrieveTree() : query;
            var deferred = When.defer(),
                executeFunc = getExecuteFunction(query);
            //console.log("IN GRAPH, EXECUTING FUNCTION", query);
            if(executeFunc) {
                this.deferred.then(function (graph) {
                    //console.log("IN GRAPH, BEFORE QUERY EXECUTION", query);
                    graph.engine.execute(query, executeFunc(deferred, {
                        callback: callback,
                        graph: graph,
                        onsuccess: onsuccess,
                        query: query
                    }));
                });
                this.deferred = deferred;
            } else {
                //console.log("IN GRAPH, QUERY NOT SUPPORTED", query);
                throw new Error("Query not supported" + query);
            }
            return this;
        },
        extend: function (resource) {
            var deferred = When.defer();
            this.deferred.then(function (graph) {
                if (resource instanceof Graph) {
                    loadGraph(graph, resource, deferred);
                } else if (Utils.isArray(resource)) {
                    if (resource[0] && Utils.isUri(resource[0])) {
                        loadUris(graph, resource, deferred);
                    } else {
                        loadStatements(graph, resource, deferred);
                    }
                } else if (Utils.isString(resource)) {
                    loadUri(graph, resource, deferred);
                } else if (resource.toNT) {
                    loadFormula(graph, resource, deferred);
                }
            });
            this.deferred = deferred;
            return this;
        },
        size: function (callback) {
            var deferred = When.defer();
            this.deferred.then(function (graph) {
                graph.engine.execute("SELECT * WHERE { ?s ?p ?o }", function (success, results) {
                    callback(results.length);
                    deferred.resolve(graph);
                });
            });
            this.deferred = deferred;
            return this;
        },
        then: function (callback) {
            this.deferred.then(callback);
        }
    };
    Graph.prototype.init.prototype = Graph.prototype;
```

```
    return Graph;
});importScripts('../../lib/require.js');

require({
        baseUrl: "./"
    },
    ["graph"],
    function (Graph) {
        self.addEventListener('message', function(e) {
            var data = e.data;
            switch (data["cmd"]) {
                case 'query':
                    self.postMessage({
                        query: data.query
                    });
                    break;
                case 'size':
                    self.postMessage("SIZE");
                    break;
                default:
                    self.postMessage("UNKNOWN COMMAND");
            }
        }, false);
    });/*global define*/
define([
    "./loader/http",
    "./loader/xhr",
    "./loader/proxy",
    "./utils"
], function (HTTP, XHR, Proxy, Utils) {
    "use strict";
    /**
     * @param {Object} options
     * <ul>
     *     <li>asynchronous: whether to load resource asynchronously or not; default set to true</li>
     *     <li>method: which HTTP verb to call resource with; default set to GET</li>
     *     <li>success: function to call if uri is successfully loaded; by default undefined. Function
     *     has parameters err, data, status, headers</li>
     *     <li>uri: adress to load resource from; by default set to document's URI</li>
     * </ul>
     */
    return function (options) {
        //console.log("IN LOADER", options);
        var support = {
            "http": typeof ModuleHttp !== "undefined",
            "xhr": typeof XMLHttpRequest !== "undefined"
        };
        options = Utils.extend({
            "followRedirect": false
        }, options);
        if(support["http"]) {
            return HTTP(options);
        } else if(support["xhr"] && options.proxy) {
            return Proxy(options);
        } else if(support["xhr"]) {
            return XHR(options);
        } else {
            throw "Your running environment doesn't support the loader";
        }
    };
});define([], function () {
    return function() {
        //TODO Implement a processor-adapter
    };
});/*global define */
define([
    "./loader",
```

```
    "./queryparser",
    "./utils",
    "./when"
], function (Loader, QueryParser, Utils, When) {
    var Query = function (queryString) {
            return new Query.prototype.init(queryString);
        },
        QueryObject = function (query, objectName) {
            return new QueryObject.prototype.init(query, objectName);
        },
        QuerySubject = function (query, subjectName) {
            return new QuerySubject.prototype.init(query, subjectName);
        },
        QueryTerm,
        sparqlRegex = /^(#|ADD|ASK|BASE|CONSTRUCT|COPY|CLEAR|CREATE|DESCRIBE|DELETE|DROP|INSERT|LOAD|MOVE|
PREFIX|SELECT|WITH)/;
    function assembleNode (node, subject, predicate, object) {
        node.query
            .where("{0} {1} {2}".format(
            subject,
            predicate,
            object
        ));
        if (node.filters.length > 0) {
            Utils.each(node.filters, function (filter) {
                node.query.filter(filter);
            }.bind(node));
        }
        if (node.regexes.length > 0) {
            Utils.each(node.regexes, function (r) {
                node.query.regex(node.termName, r.pattern, r.flags);
            }.bind(node));
        }
    }
    function cleanArg(str) {
        if (Utils.isString(str)) {
            return str.replace(/"/g, '\\"');
        }
        return str;
    }
    function findBGP(pattern) {
        var bgpindex = null;
        Utils.each(pattern.patterns, function (p, i) {
            if (p.token === "basicgraphpattern") {
                bgpindex = i;
            }
        });
        return bgpindex;
    }
    function findUnit(syntaxTree) {
        return syntaxTree.units.length - 1;
    }
    function format(queryString, args, numOfParam) {
        args = Utils.toArray(args).slice(numOfParam);
        if (args.length > 0) {
            args.push(cleanArg);
            queryString = String.prototype.format.apply(queryString, args);
        }
        //console.debug("IN QUERY, FORMAT QUERY STRING", queryString);
        return queryString;
    }
    function formatTerm(term) {
        if (term.length > 7 && term.substr(0, 7) === "http://") {
            return "<" + term + ">";
        }
        return term;
    }
```

```
    function initiateQuery(queryString, deferred) {
        //console.log("IN QUERY, initiateQuery", queryString);
        this.syntaxTree = this.parser.parse(queryString).syntaxTree;
        if (this.syntaxTree.kind === "query") {
            this.unitindex = findUnit(this.syntaxTree);
            this.pattern = this.syntaxTree.units[this.unitindex].pattern;
            this.projection = this.syntaxTree.units[this.unitindex].projection;
            this.bgpindex = findBGP(this.pattern);
            this.variables = Utils.map(this.projection, function (p) {
                if (p.kind === "*") {
                    return p.kind;
                } else if (p.kind === "aliased") {
                    return p.alias.value;
                }
                return p.value.value;
            });
        }
        deferred.resolve(true);
    }
    function tokenWhere(pattern) {
        //console.log("IN QUERY, tokenWhere", this);
        var where = this.parser.where("WHERE {" + pattern + "}", {
            bgpindex: this.bgpindex,
            pattern: this.pattern
        });
        this.bgpindex = Utils.isNumber(where.bgpindex) ? where.bgpindex : this.bgpindex;
        this.pattern = where.where;
    }
    Query.prototype = {
        init: function (queryStringOrUri) {
            var self = this,
                deferred = When.defer();
            this.prologueBase = null;
            this.bgpindex = null;
            this.pattern = null;
            this.prefixes = {};
            this.projection = null;
            this.variables = [];
            this.unitGroup = null;
            this.unitindex = 0;
            this.parser = QueryParser("sparql");
            if(queryStringOrUri && Utils.isString(queryStringOrUri) && sparqlRegex.test
(queryStringOrUri)) {
                queryStringOrUri = format(queryStringOrUri, arguments, 1);
                //console.log("IN QUERY, INIT, WITH", queryStringOrUri);
                initiateQuery.call(this, queryStringOrUri, deferred);
            } else if (queryStringOrUri && Utils.isString(queryStringOrUri)) {
                //console.log("IN QUERY, INIT, URI", queryStringOrUri);
                Loader({
                    asynchronous: false,
                    success: function (err, data) {
                        initiateQuery.call(self, data, deferred);
                    },
                    uri: queryStringOrUri
                })
            } else if (queryStringOrUri) {
                deferred.resolve(false);
                throw new Error("Query given invalid");
            } else {
                //console.log("IN QUERY, INIT, WITHOUT");
                this.syntaxTree = this.parser.parse("SELECT * WHERE { ?subject ?predicate ?
object }").syntaxTree;
                deferred.resolve(true);
            }
            this.deferred = deferred;
            return this;
        },
```

```
    base: function (value) {
        value = format(value, arguments, 1);
        this.prologueBase = this.parser.base(value).base;
        return this;
    },
    filter: function (filter) {
        filter = format(filter, arguments, 1);
        //console.log("IN QUERY, filter", filter);
        tokenWhere.call(this, "FILTER(" + filter + ")");
        return this;
    },
    getObject: function (variableName) {
        return new QueryObject(this, variableName);
    },
    getSubject: function (variableName) {
        return new QuerySubject(this, variableName);
    },
    group: function (group) {
        group = format(group, arguments, 1);
        this.unitGroup = this.parser.group(group).group;
        return this;
    },
    optional: function (optional) {
        optional = format(optional, arguments, 1);
        tokenWhere.call(this, "OPTIONAL { " + optional + " }");
        return this;
    },
    prefix: function (prefix, local) {
        prefix = format(prefix, arguments, 2);
        local = format(local, arguments, 2);
        if (!this.prefixes[prefix]) {
            var token = this.parser.prefix("{0}: <{1}>".format(prefix, local));
            this.prefixes[prefix] = token.prefix;
        }
        return this;
    },
    /**
     *
     * @param variable
     * @param pattern
     * @param [flags]
     * @return {*}
     */
    regex: function (variable, pattern, flags) {
        variable = format(variable, arguments, 3);
        pattern = format(pattern, arguments, 3);
        flags = format(flags, arguments, 3);
        tokenWhere.call(this, flags ?
            'FILTER regex({0}, "{1}", "{2}")'.format(variable, pattern, flags) :
            'FILTER regex({0}, "{1}")'.format(variable, pattern));
        return this;
    },
    retrieveTree: function () {
        if (this.prologueBase) {
            this.syntaxTree.prologue.base = this.prologueBase;
        }
        if (this.pattern) {
            this.syntaxTree.units[this.unitindex].pattern = this.pattern;
        }
        if (Utils.size(this.prefixes) > 0) {
            this.syntaxTree.prologue.prefixes = Utils.toArray(this.prefixes);
        }
        if (this.projection) {
            this.syntaxTree.units[this.unitindex].projection = this.projection;
        }
        if (this.unitGroup) {
            this.syntaxTree.units[this.unitindex].group = this.unitGroup;
```

```
            }
            //console.log("IN QUERY, retrieveTree assembled", this.syntaxTree);
            return this.syntaxTree;
        },
        select: function (projection) {
            projection = format(projection, arguments, 1);
            this.projection = this.parser.projection(projection).projection;
            return this;
        },
        Than: function (callback) {
            this.deferred.Than(callback);
            return this;
        },
        where: function (pattern) {
            pattern = format(pattern, arguments, 1);
            //console.log("IN QUERY, WHERE", this.pattern);
            tokenWhere.call(this, pattern);
            return this;
        }
    };
    Query.prototype.init.prototype = Query.prototype;
    QueryObject.prototype = {
        init: function (query, objectName) {
            this.query = query;
            this.termName = objectName;
            this.predicateName = "?predicate";
            this.subjectName = "?subject";
            this.filters = [];
            this.regexes = [];
        },
        asSubject: function (subjectName) {
            assembleNode(this, subjectName, this.predicateName, this.termName);
            return new QuerySubject(this.query, subjectName);
        },
        getSubjectAsObject: function (objectName) {
            assembleNode(this, objectName, this.predicateName, this.termName);
            return new QueryObject(this.query, objectName);
        },
        retrieveTree: function () {
            assembleNode(this, this.subjectName, this.predicateName, this.termName);
            return this.query.retrieveTree();
        },
        withSubject: function (subjectName) {
            this.subjectName = formatTerm(subjectName);
        }
    };
    QuerySubject.prototype = {
        init: function (query, subjectName) {
            this.query = query;
            this.objectName = "?object";
            this.predicateName = "?predicate";
            this.termName = subjectName;
            this.filters = [];
            this.regexes = [];
        },
        asObject: function (objectName) {
            assembleNode(this, this.termName, this.predicateName, objectName);
            return new QueryObject(this.query, objectName);
        },
        getObjectAsSubject: function (subjectName) {
            assembleNode(this, this.termName, this.predicateName, subjectName);
            return new QuerySubject(this.query, subjectName);
        },
        retrieveTree: function () {
            assembleNode(this, this.termName, this.predicateName, this.objectName);
            return this.query.retrieveTree();
        },
```

```
        withObject: function (objectName) {
            this.objectName = formatTerm(objectName);
        }
    };
    QueryTerm = {
        equals: function (value) {
            value = format(value, arguments, 1);
            this.filters.push("{0} = {1}".format(this.termName, value));
        },
        greaterThan: function (value) {
            value = format(value, arguments, 1);
            this.filters.push("{0} > {1}".format(this.termName, value));
        },
        greaterOrEqualThan: function (value) {
            value = format(value, arguments, 1);
            this.filters.push("{0} >= {1}".format(this.termName, value));
        },
        lesserThan: function (value) {
            value = format(value, arguments, 1);
            this.filters.push("{0} < {1}".format(this.termName, value));
        },
        lesserOrEqualThan: function (value) {
            value = format(value, arguments, 1);
            this.filters.push("{0} <= {1}".format(this.termName, value));
        },
        regex: function (pattern, flags) {
            pattern = format(pattern, arguments, 2);
            flags = flags ? format(flags, arguments, 2) : null;
            this.regexes.push({
                flags: flags,
                pattern: pattern
            });
        },
        withProperty: function (predicateName) {
            this.predicateName = formatTerm(predicateName);
        }
    };
    Utils.extend(QueryObject.prototype.init.prototype, QueryTerm, QueryObject.prototype);
    Utils.extend(QuerySubject.prototype.init.prototype, QueryTerm, QuerySubject.prototype);
    return Query;
});/*global define */
define([
    "./queryparser/sparql"
], function (SPARQL) {
    "use strict";
    var parsers = {
        "sparql": SPARQL
    };
    /**
     * The parser object
     * @param {String} format The type of query that is loaded
     */
    return function (format) {
        if (parsers[format]) {
            return parsers[format];
        }
        throw new Error("Format not recognized");
    };
});/*global define */
define([
    "./utils",
    "./rdfparser/jsonld",
    "./rdfparser/rdfjson",
    "../rdfquery/rdfparser/rdfxml",
    "../rdfquery/rdfparser/turtle"
], function (Utils, JSONLD, RDFJSON, RDFXML, TTL) {
    "use strict";
```

```
    var parsers = {
        "json": JSONLD,
        "jsonld": JSONLD,
        "json-ld": JSONLD,
        "rdfjson": RDFJSON,
        "rdf/json": RDFJSON,
        "rdf+json": RDFJSON,
        "rdfxml": RDFXML,
        "rdf/xml": RDFXML,
        "rdf+xml": RDFXML,
        "ttl": TTL,
        "turtle": TTL
    };
    /**
     * The parser object
     * @param {String|Object} format The data to be parsed
     * @param {String} format The MIME format of the data to be parsed
     * @param {Object} options Various options available for the parser
     * @param {Function} callback The function to call when the data is parsed; parameter is
     * the graph assembled, given in form of Dictionary.Formula
     */
    return function (data, format, options, callback) {
        //console.log("IN PARSER", format, parsers[format]);
        if (!callback && Utils.isFunction(options)) {
            callback = options;
            options = {};
        }
        if (parsers[format]) {
            parsers[format](data, options, callback);
        } else {
            throw new Error("Format not recognized");
        }
    };
});
define([
    "./serializer/sparql",
    "./utils"
], function (Sparql) {
    var formats = {
        "sparql": Sparql
    };
    return function (input, format, options) {
        if(formats[format]) {
            return formats[format](input, options);
        }
        throw "The requested format " + format + " isn't supported!";
    };
});define([
    "./../rdfstore/rdf-persistence/lexicon"
], function (Lexicon) {
    var breaker          = {},
        ArrayProto       = Array.prototype,
        ObjProto         = Object.prototype,
        FuncProto        = Function.prototype,
        nativeForEach    = ArrayProto.forEach,
        nativeBind       = FuncProto.bind,
        nativeFilter     = ArrayProto.filter,
        nativeIndexOf    = ArrayProto.indexOf,
        nativeMap        = ArrayProto.map,
        nativeReduce     = ArrayProto.reduce,
        nativeSome       = ArrayProto.some,
        slice            = ArrayProto.slice,
        unshift          = ArrayProto.unshift,
        toString         = ObjProto.toString,
        hasOwnProperty   = ObjProto.hasOwnProperty,
        nativeTrim       = String.prototype.trim,
        Utils = {},
```

```
        lexicon = new Lexicon.Lexicon();

    if (!Object.create) {
        var F;
        Object.create = function (obj) {
            F = function () {};
            F.prototype = obj;
            return new F;
        }
    }

    var defaultToWhiteSpace = function (characters){
        if (characters != null) {
            return '[' + _s.escapeRegExp(''+characters) + ']';
        }
        return '\\s';
    };
    /**
     * Determine if at least one element in the object matches a truth test.
     * Delegates to ECMAScript 5's native some if available. Aliased as any.
     * @param obj
     * @param iterator
     * @param context
     */
    Utils.any = function (obj, iterator, context) {
        iterator || (iterator = Utils.identity);
        var result = false;
        if (obj == null) return result;
        if (nativeSome && obj.some === nativeSome) return obj.some(iterator, context);
        Utils.each(obj, function (value, index, list) {
            if (result || (result = iterator.call(context, value, index, list))) return breaker;
        });
        return !!result;
    };
    /**
     * Taken from underscore.js
     *
     * Create a function bound to a given object (assigning this, and
     * arguments, optionally). Binding with arguments is also known as
     * curry. Delegates to ECMAScript 5's native Function.bind if
     * available. We check for func.bind first, to fail fast when func is
     * undefined.
     * @param func
     * @param context
     */
    Utils.bind = function bind(func, context) {
        var bound, args;
        if (func.bind === nativeBind && nativeBind) return nativeBind.apply(func, slice.call(arguments,
1));
        if (!Utils.isFunction(func)) throw new TypeError;
        args = slice.call(arguments, 2);
        return bound = function () {
            var ctor = {};
            if (!(this instanceof bound)) {
                return func.apply(context, args.concat(slice.call(arguments)));
            }
            ctor.prototype = func.prototype;
            var self = new ctor;
            var result = func.apply(self, args.concat(slice.call(arguments)));
            if (Object(result) === result) {
                return result;
            }
            return self;
        };
    };
    /**
     * Create a (shallow-cloned) duplicate of an object.
```

```
     * Taken from underscore.js
     */
    Utils.clone = function (obj) {
        if (!Utils.isObject(obj)) {
            return obj;
        }
        if (Utils.isArray(obj)) {
            return obj.slice();
        }
        var objB = {};
        Utils.each(obj, function (value, key) {
            objB[key] = Utils.clone(value);
        });
        return objB;
    };
    /**
     * Create an instance of a given object
     *
     * @param {Object} obj The object to be instanciated
     * @returns {Object} The instanciated object
     */
    Utils.create = function (obj) {
        function getArgs(arr) {
            var args = Utils.toArray(arr);
            args.shift();
            return args;
        }
        function reset(prop) {
            if (Utils.isArray(prop)) return [];
            if (Utils.isFunction(prop)) return prop;
            if (Utils.isNumber(prop)) return 0;
            return undefined;
        }

        if (Utils.isFunction(obj)) {
            obj = obj.apply({}, getArgs(arguments));
        } else {
            obj = Object.create(obj);
            if (obj.init && typeof obj.init === "function") {
                return obj.init.apply(obj, getArgs(arguments)) || obj;
            }
            for(var prop in obj) {
                obj[prop] = reset(obj[prop]);
            }
        }
        return obj;
    };
    /**
     * Takes a value and converts it to a literal accordingly to the Turtle-format
     *
     * @param literal {String|Integer} The value to check
     * @param options {Object} Looks for one of two properties, ie. lang or datatype
     * @return {String} The converted literal
     */
    Utils.createLiteral = function (literal) {
        var tmp,
            lang,
            type;
        if (Utils.isObject(literal)) {
            lang = literal.lang;
            type = literal.datatype;
            literal = '"' + literal.value + '"';
        }
        if (Utils.isString(literal)) {
            tmp = literal[0] === '"' ? literal : '"' + literal + '"';
        } else {
            tmp = '"' + literal + '"';
```

```
        }
        if (lang) {
            tmp += "@" + lang;
        } else if (type && Utils.isUri(type)) {
            tmp += "^^<" + type + ">";
        } else if (type) {
            tmp += "^^" + type;
        } else {
            type = this.getDatatype(literal);
            if (type) {
                tmp += "^^<" + type + ">";
            }
        }
        return tmp;
    };
    Utils.createResource = function (resource) {
        resource = Utils.isObject(resource) ? resource : {
            value: resource,
            token: this.getToken(resource)
        };
        return resource.token === "uri" ? '<' + resource.value + '>' : Utils.createLiteral(resource);
    };
    Utils.createTriple = function (subject, predicate, object) {
        var lang,
            datatype;
        subject = subject || {
            value: '_:' + lexicon.registerBlank(),
            token: "uri"
        };
        subject = Utils.isObject(subject) ? subject : {
            value: subject,
            token: "uri"
        };
        predicate = Utils.isObject(predicate) ? predicate : {
            value: predicate,
            token: "uri"
        };
        object = Utils.isObject(object) ? object : {
            value: object,
            token: Utils.getToken(object)
        };
        if (object.token === "literal") {
            datatype = Utils.getDatatype(object.value);
            if (datatype) {
                object.datatype = datatype;
            }
            lang = Utils.getLang(object.value);
            if (lang) {
                object.lang = lang;
            }
            object.value = Utils.getValue(object.value, datatype, lang);
        }
        return {
            subject: subject,
            predicate: predicate,
            object: object,
            statement: Utils.createResource(subject) + ' ' + Utils.createResource(predicate) + ' ' +
Utils.createResource(object) + ' .'
        };
    };
    /**
     * Taken from underscore
     *
     * Take the difference between one array and a number of other arrays.
     * Only the elements present in just the first array will remain.
     * @param array
     */
```

```
    Utils.difference = function (array) {
        var rest = Utils.flatten(slice.call(arguments, 1), true);
        return Utils.filter(array, function (value){ return !Utils.include(rest, value); });
    };
    /**
     * Handles objects with the built-in forEach, arrays, and raw objects. Delegates to ECMAScript 5's
native forEach if available.
     * Taken from underscore.js
     */
    Utils.each = function (obj, iterator, context) {
        if (obj == null) return;
        if (nativeForEach && obj.forEach === nativeForEach) {
            obj.forEach(iterator, context);
        } else if (obj.length === +obj.length) {
            for (var i = 0, l = obj.length; i < l; i++) {
                if (i in obj && iterator.call(context, obj[i], i, obj) === breaker) return;
            }
        } else {
            for (var key in obj) {
                if (Utils.has(obj, key)) {
                    if (iterator.call(context, obj[key], key, obj) === breaker) return;
                }
            }
        }
    };
    /**
     * Extend a given object with all the properties in passed-in object(s).
     * Taken from underscore.js
     */
    Utils.extend = function (obj) {
        Utils.each(slice.call(arguments, 1), function (source) {
            for (var prop in source) {
                obj[prop] = source[prop];
            }
        });
        return obj;
    };
    /**
     * Extract the value from a key from a map and delete its presence in the map
     *
     * @param {Object} map The map to extract from
     * @param {string} key The key to extract
     * @returns {Varies} The value the key pointed to
     */
    Utils.extract = function (map, key) {
        if (map.hasOwnProperty(key)) {
            var tmp = map[key];
            delete map[key];
            return tmp;
        }
    };
    /**
     * Fetched from http://bit.ly/dnsqTn
     *
     * I take the given function [as a string] and extract
     * the arguments as a named-argument map. This will
     * return the map as an array of ordered names.
     */
    Utils.extractArgumentMap = function (functionCode){
        var argumentStringMatch = functionCode.toString().match(new RegExp("\\([^)]*\\)", "")),
            argumentMap = argumentStringMatch[0].match(new RegExp("[^\\s,()]+", "g"));
        return argumentMap || [];
    };
    /**
     * Taken from underscore.js
     *
     * Return all the elements that pass a truth test. Delegates to ECMAScript
```

```
     * 5's native filter if available. Aliased as select.
     */
    Utils.filter = function (obj, iterator, context) {
        var results = [];
        if (obj == null) return results;
        if (nativeFilter && obj.filter === nativeFilter) return obj.filter(iterator, context);
        Utils.each(obj, function (value, index, list) {
            if (iterator.call(context, value, index, list)) results[results.length] = value;
        });
        return results;
    };
    /**
     * Taken from underscore.js
     *
     * Return a completely flattened version of an array.
     * @param array
     * @param [shallow]
     */
    Utils.flatten = function (array, shallow) {
        return Utils.reduce(array, function (memo, value) {
            if (Utils.isArray(value)) return memo.concat(shallow ? value : Utils.flatten(value));
            memo[memo.length] = value;
            return memo;
        }, []);
    };
    /**
     *
     * @param {String} str The string to be formatted
     * @return {String} The formatted string
     */
    String.prototype.format = function() {
        var args = Utils.toArray(arguments),
            last = Utils.last(args),
            fn = function (str) { return str; }
        if (Utils.isFunction(last)) {
            fn = last;
            args.pop();
        }
        return this.replace(/{(\d+)}/g, function(match, number) {
            return typeof args[number] != 'undefined'
                ? fn(args[number])
                : fn(match);
        });
    };
    /**
     *
     * @param literal
     * @return {*}
     */
    Utils.getDatatype = function (literal) {
        var datatypeAt;
        if (Utils.isString(literal)) {
            datatypeAt = literal.indexOf('"^^');
            if (datatypeAt > 0) {
                return literal.substring(datatypeAt + 4, literal.length - 1);
            }
            return undefined;
        } else if (Utils.isBoolean(literal)) {
            return "http://www.w3.org/2001/XMLSchema#boolean";
        } else if (Utils.isInteger(literal)) {
            return "http://www.w3.org/2001/XMLSchema#integer";
        } else if (Utils.isDouble(literal)) {
            return "http://www.w3.org/2001/XMLSchema#double";
        }
        return undefined;
    };
    Utils.getLang = function (literal) {
```

```
        var langAt;
        if (Utils.isString(literal)) {
            langAt = literal.indexOf('"@');
            if (langAt > 0) {
                return literal.substring(langAt + 2);
            }
        }
        return undefined;
    };
    /**
     * Figures whether or not the given object is a uri or a literal
     * @param object {Varies} The object to check whether is a uri or a literal
     * @returns {String} Either uri or literal
     */
    Utils.getToken = function (object) {
        return Utils.isUri(object) ? "uri" : "literal";
    };
    Utils.getTriples = function (query) {
        var tripleRegex = /<(_:[0-9]+|http:\/\/[a-zA-Z0-9#_\-.\/]+)>\s+<http:\/\/[a-zA-Z0-9#_\-.\/]+>\s
+("[a-zA-Z0-9\s\-_\/]+"\^\^<http:\/\/[a-zA-Z0-9#_\-.\/]+>|"[a-zA-Z0-9\s\-_\/]+"|<(_:[0-9]+|http:\/\/[a-zA-
Z0-9#_\-.\/]+|)>)\s*[.]?/g,
            triples = query.match(tripleRegex);
        return triples !== null ? triples : [];
    };
    /**
     * Get a URI from a CURIE and given prefixes
     *
     * @param curie The CURIE that will be processed to a URI
     * @param prefixes A given map of prefixes
     * @returns {String} A processed string
     */
    Utils.getUri = function (curie, prefixes) {
        if (!curie) {
            return undefined;
        }
        curie = Utils.isString(prefixes[curie]) ? prefixes[curie] : curie;
        var isCurie = curie.match(/^[a-zA-Z]+:/),
            prefix,
            suffix;
        if (!isCurie || Utils.isUri(curie)) {
            return curie;
        }
        prefix = isCurie[0].replace(":", "");
        prefix = prefixes[prefix] || prefix + ":";
        suffix = curie.match(/^[a-zA-Z]+:([a-zA-Z]+)/);
        suffix = suffix ? suffix[1] : "";
        return prefix + suffix;
    };
    Utils.getValue = function (literal, datatype, lang) {
        if (!Utils.isString(literal)) {
            return literal;
        }
        if (datatype) {
            return literal.substring(1, literal.length - datatype.length - 5);
        } else if (lang) {
            return literal.substring(1, literal.length - lang.length - 2);
        }
        return literal;
    };
    /**
     * Has own property?
     * Taken from underscore.js
     */
    Utils.has = function (obj, key) {
        return hasOwnProperty.call(obj, key);
    };
    /**
```

```
     * Keep the identity function around for default iterators.
     * Taken from underscore.js
     */
    Utils.identity = function (value) {
        return value;
    };
    /**
     * Taken from underscore
     *
     * Determine if a given value is included in the array or object using ===.
     * Aliased as contains.
     */
    Utils.include = function (obj, target) {
        var found = false;
        if (obj == null) return found;
        if (nativeIndexOf && obj.indexOf === nativeIndexOf) return obj.indexOf(target) != -1;
        found = Utils.any(obj, function (value) {
            return value === target;
        });
        return found;
    };
    /**
     * Return the position of the first occurrence of an item in an array, or -1 if the item is not
included in the array.
     * Delegates to ECMAScript 5's native indexOf if available.
     * If the array is large and already in sort order, pass true for isSorted to use binary search.
     * Taken from underscore.js
     */
    Utils.indexOf = function (array, item, isSorted) {
        if (array == null) return -1;
        var i, l;
        if (isSorted) {
            i = Utils.sortedIndex(array, item);
            return array[i] === item ? i : -1;
        }
        if (nativeIndexOf && array.indexOf === nativeIndexOf) return array.indexOf(item);
        for (i = 0, l = array.length; i < l; i++) {
            if (array[i] && array[i] === item) {
                return i;
            }
        }
        return -1;
    };
    /**
     * Is a given variable an arguments object?
     * Taken from underscore.js
     *
     * @param {Object} obj The object to check for arguments
     * @returns {Boolean} True if the object is an arguments object
     */
    Utils.isArguments = function (obj) {
        return toString.call(obj) == '[object Arguments]';
    };
    if (!Utils.isArguments(arguments)) {
        Utils.isArguments = function (obj) {
            return !!(obj && Utils.has(obj, 'callee'));
        };
    }
    /**
     * Determine if an object is an `Array`.
     *
     * @param {Object} object An object that may or may not be an array
     * @returns {Boolean} True if the parameter is an array
     */
    Utils.isArray = Array.isArray || function (object) {
        return !!(object && object.concat
            && object.unshift && !object.callee);
```

```
};
/**
 * Is a given value a boolean?
 * Taken from underscore.js
 *
 * @param {Varies} obj The object to test
 * @returns {Boolean} True if the object is a boolean
 */
Utils.isBoolean = function (obj) {
    return obj === true || obj === false || toString.call(obj) == '[object Boolean]';
};
/**
 * Determines if the object is a double/float
 *
 * @param {Varies} obj A value to test
 * @returns {Boolean} True if the value is a double/float
 */
Utils.isDouble = function (obj) {
    return !Utils.isNaN(parseFloat(obj));
};
/**
 * Determines if an object is a `Function`.
 *
 * @param {Object} object A value to test
 * @returns {Boolean} True if the object is a Function
 */
Utils.isFunction = function (object) {
    return typeof object === "function";
};
/**
 * Determines if an object is an integer.
 *
 * @param {Object} obj A value to test
 * @returns {Boolean} True if the object is an integer
 */
Utils.isInteger = function (obj) {
    return (parseFloat(obj) == parseInt(obj)) && !Utils.isNaN(obj);
};
/**
 * Is the given value NaN?
 * NaN is the only value for which === is not reflexive.
 * Taken from underscore.js
 *
 * @param {Varies} obj The object to test
 * @returns {Boolean} True if the object is a decimal
 */
Utils.isNaN = function (obj) {
    return obj !== obj;
};
/**
 * Determines if an object is a `Number`.
 *
 * @param {Object} object A value to test
 * @returns {Boolean} True if the object is a Number
 */
Utils.isNumber = function (object) {
    return (object === +object) || (toString.call(object) === '[object Number]');
};
/**
 * Is a given variable an object?
 * Taken from underscore.js
 */
Utils.isObject = function (obj) {
    return obj === Object(obj);
};
/**
 * Determines if an object is a `String`.
```

```
     *
     * @param {Object} object A value to test
     * @return {Boolean} True if the object is a String
     */
    Utils.isString = function (object) {
        return typeof object === "string";
    };
    /**
     * Determines if an object is a `Uri`.
     *
     * @param {Object} object A value to test
     * @return {Boolean} True is the object is a Uri
     */
    Utils.isUri = function (object) {
        return Utils.isString(object) && Utils.isArray(object.match(/^http:/));
    };
    /**
     * Return the results of applying the iterator to each element. Delegates to ECMAScript 5's native
map if available.
     * Taken from underscore.js
     */
    Utils.map = function (obj, iterator, context) {
        var results = [];
        if (obj == null) {
            return results;
        }
        if (nativeMap && obj.map === nativeMap) {
            return obj.map(iterator, context);
        }
        Utils.each(obj, function (value, index, list) {
            results[results.length] = iterator.call(context, value, index, list);
        });
        if (obj.length === +obj.length) results.length = obj.length;
        return results;
    };
    Utils.mapArgs = function (resultMap, argumentsMap) {
        //console.log("IN UTILS, MAP ARGS", resultMap, argumentsMap);
        return Utils.map(resultMap, function (arg) {
            if (argumentsMap.hasOwnProperty(arg)) {
                return argumentsMap[arg];
            }
            throw new Error ("Argument not in projection " + arg);
        });
    };
    Utils.mapFunctionArgs = function (functionCode, argumentsMap) {
        var functionMap = Utils.extractArgumentMap(functionCode);
        return Utils.mapArgs(functionMap, argumentsMap);
    };
    /**
     * Parse a URI according to http://tools.ietf.org/html/rfc3986
     * parseUri 1.2.2
     * (c) Steven Levithan <stevenlevithan.com>
     *
     * @param {String} str The string to parse
     * @returns {Object} The parsed string, as an object (null if an invalid uri is passed)
     */
    Utils.parseUri = function (str) {
        var o = {
            strictMode: false,
            key:
["source","protocol","authority","userInfo","user","password","host","port","relative","path","directory","fil
            q:   {
                name:   "queryKey",
                parser: /(?:^|&)([^&=]*)=?([^&]*)/g
            },
            parser: {
                strict: /^(?:([^:\/?#]+):)?(?:\/\/((?:(([^:@]*)(?::([^:@]*))?)?@)?([^:\/?#]*)(?::(\d*))?
```

```
))?((((?:[^?#\/]*\/)*)([^?#]*))(?:\?([^#]*))?(?:#(.*))?)/,
                loose:  /^(?:(?![^:@]+:[^:@\/]*@)([^:\/?#.]+):)?(?:\/\/)?((?:(([^:@]*)(?::([^:@]*))?)?@)?
([^:\/?#]*)(?::(\d*))?)(((\/(?:[^?#](?![^?#\/]*\.[^?#\/.]+(?:[?#]|$)))*\/?)?([^?#\/]*))(?:\?([^#]*))?(?:#
(.*))?)/
            }
            },
            m = o.parser[o.strictMode ? "strict" : "loose"].exec(str),
            uri = {},
            i   = 14;
        while (i--) uri[o.key[i]] = m[i] || "";
        uri[o.q.name] = {};
        uri[o.key[12]].replace(o.q.parser, function ($0, $1, $2) {
            if ($1) uri[o.q.name][$1] = $2;
        });
        return uri;
    };
    /**
     * Taken from underscore.js
     *
     * Get the last element of an array. Passing n will return the last N
     * values in the array. The guard check allows it to work with Utils.map.
     *
     * @param array
     * @param [n]
     * @param [guard]
     * @return {*}
     */
    Utils.last = function(array, n, guard) {
        if ((n != null) && !guard) {
            return slice.call(array, Math.max(array.length - n, 0));
        } else {
            return array[array.length - 1];
        }
    };
    /**
     * Taken from underscore.js
     *
     * Reduce builds up a single result from a list of values, aka inject, or
     * foldl. Delegates to ECMAScript 5's native reduce if available.
     * @param obj
     * @param iterator
     * @param memo
     * @param context
     */
    Utils.reduce = function (obj, iterator, memo, context) {
        var initial = arguments.length > 2;
        if (obj == null) obj = [];
        if (nativeReduce && obj.reduce === nativeReduce) {
            if (context) iterator = Utils.bind(iterator, context);
            return initial ? obj.reduce(iterator, memo) : obj.reduce(iterator);
        }
        Utils.each(obj, function (value, index, list) {
            if (!initial) {
                memo = value;
                initial = true;
            } else {
                memo = iterator.call(context, memo, value, index, list);
            }
        });
        if (!initial) throw new TypeError('Reduce of empty array with no initial value');
        return memo;
    };
    /**
     * Return the number of elements in an object.
     * Taken from underscore.js
     *
     * @param {Object} obj The object to determine the size of
```

```
     * @returns {Number} The size of the object
     */
    Utils.size = function (obj) {
        return Utils.toArray(obj).length;
    };

    /**
     * Use a comparator function to figure out at what index an object should be inserted so as to
maintain order. Uses binary search.
     * Taken from underscore.js
     */
    Utils.sortedIndex = function (array, obj, iterator) {
        iterator || (iterator = Utils.identity);
        var low = 0, high = array.length;
        while (low < high) {
            var mid = (low + high) >> 1;
            iterator(array[mid]) < iterator(obj) ? low = mid + 1 : high = mid;
        }
        return low;
    };

    /**
     * Safely convert anything iterable into a real, live array.
     * Taken from underscore.js
     *
     * @param {Varies} iterable The object to convert to an array
     * @returns {Array} The converted object
     */
    Utils.toArray = function (iterable) {
        if (!iterable)                      return [];
        if (iterable.toArray)               return iterable.toArray();
        if (Utils.isArray(iterable))      return slice.call(iterable);
        if (Utils.isArguments(iterable))  return slice.call(iterable);
        return Utils.values(iterable);
    };

    Utils.trim = function (str, characters){
        str = ''+str;
        if (!characters && nativeTrim) {
            return nativeTrim.call(str);
        }
        characters = defaultToWhiteSpace(characters);
        return str.replace(new RegExp('\^' + characters + '+|' + characters + '+$', 'g'), '');
    };
    /**
     * Taken from underscore.js
     *
     * Produce a duplicate-free version of the array. If the array has already been sorted, you
     * have the option of using a faster algorithm. Aliased as unique.
     *
     * The isSorted flag is irrelevant if the array only contains two elements.
     *
     * @param array
     * @param isSorted
     * @param iterator
     * @return {Array}
     */
    Utils.unique = function(array, isSorted, iterator) {
        var initial = iterator ? Utils.map(array, iterator) : array;
        var results = [];
        if (array.length < 3) isSorted = true;
        Utils.reduce(initial, function (memo, value, index) {
            if (isSorted ? Utils.last(memo) !== value || !memo.length : !Utils.include(memo, value)) {
                memo.push(value);
                results.push(array[index]);
            }
            return memo;
```

```
        }, []);
        return results;
    };
    /**
     * Retrieve the values of an object's properties.
     * Taken from underscore.js
     *
     * @param {Object} obj The object to retrieve values from
     * @returns {Object} The retrieved values
     */
    Utils.values = function (obj) {
        return Utils.map(obj, Utils.identity);
    };
    /**
     * Taken from underscore
     *
     * Return a version of the array that does not contain the specified
     * value(s).
     * @param array
     */
    Utils.without = function (array) {
        return Utils.difference(array, slice.call(arguments, 1));
    }
    return Utils;
});/** @license MIT License (c) copyright B Cavalier & J Hann */

/**
 * when
 * A lightweight CommonJS Promises/A and when() implementation
 *
 * when is part of the cujo.js family of libraries (http://cujojs.com/)
 *
 * Licensed under the MIT License at:
 * http://www.opensource.org/licenses/mit-license.php
 *
 * @version 1.2.0
 */

define([], function() {
    var freeze, reduceArray, slice, undef;

    //
    // Public API
    //

    when.defer     = defer;
    when.reject    = reject;
    when.isPromise = isPromise;

    when.all       = all;
    when.some      = some;
    when.any       = any;

    when.map       = map;
    when.reduce    = reduce;

    when.chain     = chain;

    /** Object.freeze */
    freeze = Object.freeze || function(o) { return o; };

    /**
     * Trusted Promise constructor.  A Promise created from this constructor is
     * a trusted when.js promise.  Any other duck-typed promise is considered
     * untrusted.
     *
     * @constructor
```

```
 */
function Promise() {}

Promise.prototype = freeze({
    always: function(alwaysback, progback) {
        return this.then(alwaysback, alwaysback, progback);
    },

    otherwise: function(errback) {
        return this.then(undef, errback);
    }
});

/**
 * Create an already-resolved promise for the supplied value
 * @private
 *
 * @param value anything
 * @return {Promise}
 */
function resolved(value) {

    var p = new Promise();

    p.then = function(callback) {
        var nextValue;
        try {
            if(callback) nextValue = callback(value);
            return promise(nextValue === undef ? value : nextValue);
        } catch(e) {
            return rejected(e);
        }
    };

    return freeze(p);
}

/**
 * Create an already-rejected {@link Promise} with the supplied
 * rejection reason.
 * @private
 *
 * @param reason rejection reason
 * @return {Promise}
 */
function rejected(reason) {

    var p = new Promise();

    p.then = function(callback, errback) {
        var nextValue;
        try {
            if(errback) {
                nextValue = errback(reason);
                return promise(nextValue === undef ? reason : nextValue)
            }

            return rejected(reason);

        } catch(e) {
            return rejected(e);
        }
    };

    return freeze(p);
}
```

```
    /**
     * Returns a rejected promise for the supplied promiseOrValue. If
     * promiseOrValue is a value, it will be the rejection value of the
     * returned promise.  If promiseOrValue is a promise, its
     * completion value will be the rejected value of the returned promise
     *
     * @param promiseOrValue {*} the rejected value of the returned {@link Promise}
     *
     * @return {Promise} rejected {@link Promise}
     */
    function reject(promiseOrValue) {
        return when(promiseOrValue, function(value) {
            return rejected(value);
        });
    }

    /**
     * Creates a new, CommonJS compliant, Deferred with fully isolated
     * resolver and promise parts, either or both of which may be given out
     * safely to consumers.
     * The Deferred itself has the full API: resolve, reject, progress, and
     * then. The resolver has resolve, reject, and progress.  The promise
     * only has then.
     *
     * @memberOf when
     * @function
     *
     * @returns {Deferred}
     */
    function defer() {
        var deferred, promise, listeners, progressHandlers, _then, _progress, complete;

        listeners = [];
        progressHandlers = [];

        /**
         * Pre-resolution then() that adds the supplied callback, errback, and progback
         * functions to the registered listeners
         *
         * @private
         *
         * @param [callback] {Function} resolution handler
         * @param [errback] {Function} rejection handler
         * @param [progback] {Function} progress handler
         *
         * @throws {Error} if any argument is not null, undefined, or a Function
         */
        _then = function unresolvedThen(callback, errback, progback) {
            var deferred = defer();

            listeners.push(function(promise) {
                promise.then(callback, errback)
                    .then(deferred.resolve, deferred.reject, deferred.progress);
            });

            progback && progressHandlers.push(progback);

            return deferred.promise;
        };

        /**
         * Registers a handler for this {@link Deferred}'s {@link Promise}.  Even though all arguments
         * are optional, each argument that *is* supplied must be null, undefined, or a Function.
         * Any other value will cause an Error to be thrown.
         *
         * @memberOf Promise
         *
```

```
     * @param [callback] {Function} resolution handler
     * @param [errback] {Function} rejection handler
     * @param [progback] {Function} progress handler
     *
     * @throws {Error} if any argument is not null, undefined, or a Function
     */
    function then(callback, errback, progback) {
        return _then(callback, errback, progback);
    }

    /**
     * Resolves this {@link Deferred}'s {@link Promise} with val as the
     * resolution value.
     *
     * @memberOf Resolver
     *
     * @param val anything
     */
    function resolve(val) {
        complete(resolved(val));
    }

    /**
     * Rejects this {@link Deferred}'s {@link Promise} with err as the
     * reason.
     *
     * @memberOf Resolver
     *
     * @param err anything
     */
    function reject(err) {
        complete(rejected(err));
    }

    /**
     * @private
     * @param update
     */
    _progress = function(update) {
        var progress, i = 0;
        while (progress = progressHandlers[i++]) progress(update);
    };

    /**
     * Emits a progress update to all progress observers registered with
     * this {@link Deferred}'s {@link Promise}
     *
     * @memberOf Resolver
     *
     * @param update anything
     */
    function progress(update) {
        _progress(update);
    }

    /**
     * Transition from pre-resolution state to post-resolution state, notifying
     * all listeners of the resolution or rejection
     *
     * @private
     *
     * @param completed {Promise} the completed value of this deferred
     */
    complete = function(completed) {
        var listener, i = 0;

        // Replace _then with one that directly notifies with the result.
```

```
            _then = completed.then;

            // Replace complete so that this Deferred can only be completed
            // once. Also Replace _progress, so that subsequent attempts to issue
            // progress throw.
            complete = _progress = function alreadyCompleted() {
                // TODO: Consider silently returning here so that parties who
                // have a reference to the resolver cannot tell that the promise
                // has been resolved using try/catch
                throw new Error("already completed");
            };

            // Free progressHandlers array since we'll never issue progress events
            // for this promise again now that it's completed
            progressHandlers = undef;

            // Notify listeners
            // Traverse all listeners registered directly with this Deferred

            while (listener = listeners[i++]) {
                listener(completed);
            }

            listeners = [];
        };

        /**
         * The full Deferred object, with both {@link Promise} and {@link Resolver}
         * parts
         * @class Deferred
         * @name Deferred
         */
        deferred = {};

        // Promise and Resolver parts
        // Freeze Promise and Resolver APIs

        promise = new Promise();
        promise.then = deferred.then = then;

        /**
         * The {@link Promise} for this {@link Deferred}
         * @memberOf Deferred
         * @name promise
         * @type {Promise}
         */
        deferred.promise = freeze(promise);

        /**
         * The {@link Resolver} for this {@link Deferred}
         * @memberOf Deferred
         * @name resolver
         * @class Resolver
         */
        deferred.resolver = freeze({
            resolve:  (deferred.resolve  = resolve),
            reject:   (deferred.reject   = reject),
            progress: (deferred.progress = progress)
        });

        return deferred;
    }

    /**
     * Determines if promiseOrValue is a promise or not.  Uses the feature
     * test from http://wiki.commonjs.org/wiki/Promises/A to determine if
     * promiseOrValue is a promise.
```

```
     *
     * @param promiseOrValue anything
     *
     * @returns {Boolean} true if promiseOrValue is a {@link Promise}
     */
    function isPromise(promiseOrValue) {
        return promiseOrValue && typeof promiseOrValue.then === 'function';
    }

    /**
     * Register an observer for a promise or immediate value.
     *
     * @function
     * @name when
     * @namespace
     *
     * @param promiseOrValue anything
     * @param {Function} [callback] callback to be called when promiseOrValue is
     *   successfully resolved.  If promiseOrValue is an immediate value, callback
     *   will be invoked immediately.
     * @param {Function} [errback] callback to be called when promiseOrValue is
     *   rejected.
     * @param {Function} [progressHandler] callback to be called when progress updates
     *   are issued for promiseOrValue.
     *
     * @returns {Promise} a new {@link Promise} that will complete with the return
     *   value of callback or errback or the completion value of promiseOrValue if
     *   callback and/or errback is not supplied.
     */
    function when(promiseOrValue, callback, errback, progressHandler) {
        // Get a promise for the input promiseOrValue
        // See promise()
        var trustedPromise = promise(promiseOrValue);

        // Register promise handlers
        return trustedPromise.then(callback, errback, progressHandler);
    }

    /**
     * Returns promiseOrValue if promiseOrValue is a {@link Promise}, a new Promise if
     * promiseOrValue is a foreign promise, or a new, already-resolved {@link Promise}
     * whose resolution value is promiseOrValue if promiseOrValue is an immediate value.
     *
     * Note that this function is not safe to export since it will return its
     * input when promiseOrValue is a {@link Promise}
     *
     * @private
     *
     * @param promiseOrValue anything
     *
     * @returns Guaranteed to return a trusted Promise.  If promiseOrValue is a when.js {@link Promise}
     *   returns promiseOrValue, otherwise, returns a new, already-resolved, when.js {@link Promise}
     *   whose resolution value is:
     *   * the resolution value of promiseOrValue if it's a foreign promise, or
     *   * promiseOrValue if it's a value
     */
    function promise(promiseOrValue) {
        var promise, deferred;

        if(promiseOrValue instanceof Promise) {
            // It's a when.js promise, so we trust it
            promise = promiseOrValue;

        } else {
            // It's not a when.js promise.  Check to see if it's a foreign promise
            // or a value.
```

```
            deferred = defer();
            if(isPromise(promiseOrValue)) {
                // It's a compliant promise, but we don't know where it came from,
                // so we don't trust its implementation entirely.  Introduce a trusted
                // middleman when.js promise

                // IMPORTANT: This is the only place when.js should ever call .then() on
                // an untrusted promise.
                promiseOrValue.then(deferred.resolve, deferred.reject, deferred.progress);
                promise = deferred.promise;

            } else {
                // It's a value, not a promise.  Create an already-resolved promise
                // for it.
                deferred.resolve(promiseOrValue);
                promise = deferred.promise;
            }
        }

        return promise;
    }

    /**
     * Return a promise that will resolve when howMany of the supplied promisesOrValues
     * have resolved. The resolution value of the returned promise will be an array of
     * length howMany containing the resolutions values of the triggering promisesOrValues.
     *
     * @memberOf when
     *
     * @param promisesOrValues {Array} array of anything, may contain a mix
     *      of {@link Promise}s and values
     * @param howMany
     * @param [callback]
     * @param [errback]
     * @param [progressHandler]
     *
     * @returns {Promise}
     */
    function some(promisesOrValues, howMany, callback, errback, progressHandler) {

        checkCallbacks(2, arguments);

        return when(promisesOrValues, function(promisesOrValues) {

            var toResolve, results, ret, deferred, resolver, rejecter, handleProgress, len, i;

            len = promisesOrValues.length >>> 0;

            toResolve = Math.max(0, Math.min(howMany, len));
            results = [];
            deferred = defer();
            ret = when(deferred, callback, errback, progressHandler);

            // Wrapper so that resolver can be replaced
            function resolve(val) {
                resolver(val);
            }

            // Wrapper so that rejecter can be replaced
            function reject(err) {
                rejecter(err);
            }

            // Wrapper so that progress can be replaced
            function progress(update) {
                handleProgress(update);
            }
```

```
        function complete() {
            resolver = rejecter = handleProgress = noop;
        }

        // No items in the input, resolve immediately
        if (!toResolve) {
            deferred.resolve(results);

        } else {
            // Resolver for promises.  Captures the value and resolves
            // the returned promise when toResolve reaches zero.
            // Overwrites resolver var with a noop once promise has
            // be resolved to cover case where n < promises.length
            resolver = function(val) {
                // This orders the values based on promise resolution order
                // Another strategy would be to use the original position of
                // the corresponding promise.
                results.push(val);

                if (!--toResolve) {
                    complete();
                    deferred.resolve(results);
                }
            };

            // Rejecter for promises.  Rejects returned promise
            // immediately, and overwrites rejecter var with a noop
            // once promise to cover case where n < promises.length.
            // TODO: Consider rejecting only when N (or promises.length - N?)
            // promises have been rejected instead of only one?
            rejecter = function(err) {
                complete();
                deferred.reject(err);
            };

            handleProgress = deferred.progress;

            // TODO: Replace while with forEach
            for(i = 0; i < len; ++i) {
                if(i in promisesOrValues) {
                    when(promisesOrValues[i], resolve, reject, progress);
                }
            }
        }

        return ret;
    });
}

/**
 * Return a promise that will resolve only once all the supplied promisesOrValues
 * have resolved. The resolution value of the returned promise will be an array
 * containing the resolution values of each of the promisesOrValues.
 *
 * @memberOf when
 *
 * @param promisesOrValues {Array|Promise} array of anything, may contain a mix
 *      of {@link Promise}s and values
 * @param [callback] {Function}
 * @param [errback] {Function}
 * @param [progressHandler] {Function}
 *
 * @returns {Promise}
 */
function all(promisesOrValues, callback, errback, progressHandler) {
```

```
        checkCallbacks(1, arguments);

        return when(promisesOrValues, function(promisesOrValues) {
            return _reduce(promisesOrValues, reduceIntoArray, []);
        }).then(callback, errback, progressHandler);
    }

    function reduceIntoArray(current, val, i) {
        current[i] = val;
        return current;
    }

    /**
     * Return a promise that will resolve when any one of the supplied promisesOrValues
     * has resolved. The resolution value of the returned promise will be the resolution
     * value of the triggering promiseOrValue.
     *
     * @memberOf when
     *
     * @param promisesOrValues {Array|Promise} array of anything, may contain a mix
     *      of {@link Promise}s and values
     * @param [callback] {Function}
     * @param [errback] {Function}
     * @param [progressHandler] {Function}
     *
     * @returns {Promise}
     */
    function any(promisesOrValues, callback, errback, progressHandler) {

        function unwrapSingleResult(val) {
            return callback ? callback(val[0]) : val[0];
        }

        return some(promisesOrValues, 1, unwrapSingleResult, errback, progressHandler);
    }

    /**
     * Traditional map function, similar to `Array.prototype.map()`, but allows
     * input to contain {@link Promise}s and/or values, and mapFunc may return
     * either a value or a {@link Promise}
     *
     * @memberOf when
     *
     * @param promise {Array|Promise} array of anything, may contain a mix
     *      of {@link Promise}s and values
     * @param mapFunc {Function} mapping function mapFunc(value) which may return
     *      either a {@link Promise} or value
     *
     * @returns {Promise} a {@link Promise} that will resolve to an array containing
     *      the mapped output values.
     */
    function map(promise, mapFunc) {
        return when(promise, function(array) {
            return _map(array, mapFunc);
        });
    }

    /**
     * Private map helper to map an array of promises
     * @private
     *
     * @param promisesOrValues {Array}
     * @param mapFunc {Function}
     * @return {Promise}
     */
    function _map(promisesOrValues, mapFunc) {
```

```
        var results, len, i;

        // Since we know the resulting length, we can preallocate the results
        // array to avoid array expansions.
        len = promisesOrValues.length >>> 0;
        results = new Array(len);

        // Since mapFunc may be async, get all invocations of it into flight
        // asap, and then use reduce() to collect all the results
        for(i = 0; i < len; i++) {
            if(i in promisesOrValues)
                results[i] = when(promisesOrValues[i], mapFunc);
        }

        // Could use all() here, but that would result in another array
        // being allocated, i.e. map() would end up allocating 2 arrays
        // of size len instead of just 1.  Since all() uses reduce()
        // anyway, avoid the additional allocation by calling reduce
        // directly.
        return _reduce(results, reduceIntoArray, results);
    }

    /**
     * Traditional reduce function, similar to `Array.prototype.reduce()`, but
     * input may contain {@link Promise}s and/or values, and reduceFunc
     * may return either a value or a {@link Promise}, *and* initialValue may
     * be a {@link Promise} for the starting value.
     *
     * @memberOf when
     *
     * @param promise {Array|Promise} array of anything, may contain a mix
     *      of {@link Promise}s and values.  May also be a {@link Promise} for
     *      an array.
     * @param reduceFunc {Function} reduce function reduce(currentValue, nextValue, index, total),
     *      where total is the total number of items being reduced, and will be the same
     *      in each call to reduceFunc.
     * @param initialValue starting value, or a {@link Promise} for the starting value
     *
     * @returns {Promise} that will resolve to the final reduced value
     */
    function reduce(promise, reduceFunc, initialValue) {
        var args = slice.call(arguments, 1);
        return when(promise, function(array) {
            return _reduce.apply(undef, [array].concat(args));
        });
    }

    /**
     * Private reduce to reduce an array of promises
     * @private
     *
     * @param promisesOrValues {Array}
     * @param reduceFunc {Function}
     * @param initialValue {*}
     * @return {Promise}
     */
    function _reduce(promisesOrValues, reduceFunc, initialValue) {

        var total, args;

        total = promisesOrValues.length;

        // Skip promisesOrValues, since it will be used as 'this' in the call
        // to the actual reduce engine below.

        // Wrap the supplied reduceFunc with one that handles promises and then
        // delegates to the supplied.
```

```
        args = [
            function (current, val, i) {
                return when(current, function (c) {
                    return when(val, function (value) {
                        return reduceFunc(c, value, i, total);
                    });
                });
            }
        ];

        if (arguments.length > 2) args.push(initialValue);

        return reduceArray.apply(promisesOrValues, args);
    }

    /**
     * Ensure that resolution of promiseOrValue will complete resolver with the completion
     * value of promiseOrValue, or instead with resolveValue if it is provided.
     *
     * @memberOf when
     *
     * @param promiseOrValue
     * @param resolver {Resolver}
     * @param [resolveValue] anything
     *
     * @returns {Promise}
     */
    function chain(promiseOrValue, resolver, resolveValue) {
        var useResolveValue = arguments.length > 2;

        return when(promiseOrValue,
            function(val) {
                if(useResolveValue) val = resolveValue;
                resolver.resolve(val);
                return val;
            },
            function(e) {
                resolver.reject(e);
                return rejected(e);
            },
            resolver.progress
        );
    }

    //
    // Utility functions
    //

    /**
     * Helper that checks arrayOfCallbacks to ensure that each element is either
     * a function, or null or undefined.
     *
     * @private
     *
     * @param arrayOfCallbacks {Array} array to check
     * @throws {Error} if any element of arrayOfCallbacks is something other than
     * a Functions, null, or undefined.
     */
    function checkCallbacks(start, arrayOfCallbacks) {
        var arg, i = arrayOfCallbacks.length;
        while(i > start) {
            arg = arrayOfCallbacks[--i];
            if (arg != null && typeof arg != 'function') throw new Error('callback is not a function');
        }
    }
```

```
    /**
     * No-Op function used in method replacement
     * @private
     */
    function noop() {}

    slice = [].slice;

    // ES5 reduce implementation if native not available
    // See: http://es5.github.com/#x15.4.4.21 as there are many
    // specifics and edge cases.
    reduceArray = [].reduce ||
        function(reduceFunc /*, initialValue */) {
            // ES5 dictates that reduce.length === 1

            // This implementation deviates from ES5 spec in the following ways:
            // 1. It does not check if reduceFunc is a Callable

            var arr, args, reduced, len, i;

            i = 0;
            arr = Object(this);
            len = arr.length >>> 0;
            args = arguments;

            // If no initialValue, use first item of array (we know length !== 0 here)
            // and adjust i to start at second item
            if(args.length <= 1) {
                // Skip to the first real element in the array
                for(;;) {
                    if(i in arr) {
                        reduced = arr[i++];
                        break;
                    }

                    // If we reached the end of the array without finding any real
                    // elements, it's a TypeError
                    if(++i >= len) {
                        throw new TypeError();
                    }
                }
            } else {
                // If initialValue provided, use it
                reduced = args[1];
            }

            // Do the actual reduce
            for(;i < len; ++i) {
                // Skip holes
                if(i in arr)
                    reduced = reduceFunc(reduced, arr[i], i, arr);
            }

            return reduced;
        };

    return when;
});define([
    "./uri",
    "./../graphite/utils"
], function (URI, Utils) {
    /*
     * jQuery CURIE @VERSION
     *
     * Copyright (c) 2008,2009 Jeni Tennison
     * Licensed under the MIT (MIT-LICENSE.txt)
     *
```

```
     * Depends:
     *  jquery.uri.js
     *  jquery.xmlns.js
     *
     * @fileOverview jQuery CURIE handling
     * @author <a href="mailto:jeni@jenitennison.com">Jeni Tennison</a>
     * @copyright (c) 2008,2009 Jeni Tennison
     * @license MIT license (MIT-LICENSE.txt)
     * @version 1.0
     * @requires jquery.uri.js
     * @requires jquery.xmlns.js
     *
     * Creates a {@link jQuery.uri} object by parsing a CURIE.
     * @methodOf jQuery
     * @param {String} curie The CURIE to be parsed
     * @param {String} uri The URI string to be converted to a CURIE.
     * @param {Object} [options] CURIE parsing options
     * @param {string} [options.reservedNamespace='http://www.w3.org/1999/xhtml/vocab#'] The namespace to
apply to a CURIE that has no prefix and either starts with a colon or is in the list of reserved local
names
     * @param {string} [options.defaultNamespace]  The namespace to apply to a CURIE with no prefix which
is not mapped to the reserved namespace by the rules given above.
     * @param {Object} [options.namespaces] A map of namespace bindings used to map CURIE prefixes to
URIs.
     * @param {string[]} [options.reserved=['alternate', 'appendix', 'bookmark', 'cite', 'chapter',
'contents', 'copyright', 'first', 'glossary', 'help', 'icon', 'index', 'last', 'license', 'meta', 'next',
'p3pv1', 'prev', 'role', 'section', 'stylesheet', 'subsection', 'start', 'top', 'up']] A list of local
names that will always be mapped to the URI specified by reservedNamespace.
     * @param {string} [options.charcase='lower'] Specifies whether the curie's case is altered before
it's interpreted. Acceptable values are:
     * <dl>
     * <dt>lower</dt><dd>Force the CURIE string to lower case.</dd>
     * <dt>upper</dt><dd>Force the CURIE string to upper case.</dd>
     * <dt>preserve</dt><dd>Preserve the original case of the CURIE. Note that this might not be possible
if the CURIE has been taken from an HTML attribute value because of the case conversions performed
automatically by browsers. For this reason, it's a good idea to avoid mixed-case CURIEs within RDFa.</dd>
     * </dl>
     * @returns {jQuery.uri} A new {@link jQuery.uri} object representing the full absolute URI specified
by the CURIE.
     */
    var Curie = function (curie, options) {
        var opts = Utils.extend({}, Curie.defaults, options || {}),
            m = /^(([^:]*):)?(.+)$/.exec(curie),
            prefix = m[2],
            local = m[3],
            ns = opts.namespaces[prefix];
        if (/^:.+/.test(curie)) { // This is the case of a CURIE like ":test"
            if (opts.reservedNamespace === undefined || opts.reservedNamespace === null) {
                throw "Malformed CURIE: No prefix and no default namespace for unprefixed CURIE " + curie;
            } else {
                ns = opts.reservedNamespace;
            }
        } else if (prefix) {
            if (ns === undefined) {
                throw "Malformed CURIE: No namespace binding for " + prefix + " in CURIE " + curie;
            }
        } else {
            if (opts.charcase === 'lower') {
                curie = curie.toLowerCase();
            } else if (opts.charcase === 'upper') {
                curie = curie.toUpperCase();
            }
            if (opts.reserved.length && Utils.indexOf(curie, opts.reserved) >= 0) {
                ns = opts.reservedNamespace;
                local = curie;
            } else if (opts.defaultNamespace === undefined || opts.defaultNamespace === null) {
                // the default namespace is provided by the application; it's not clear whether
```

```
                // the default XML namespace should be used if there's a colon but no prefix
                throw "Malformed CURIE: No prefix and no default namespace for unprefixed CURIE " + curie;
            } else {
                ns = opts.defaultNamespace;
            }
        }
        return URI(ns + local);
    };

    Curie.defaults = {
        namespaces: {
            xhv: "http://www.w3.org/1999/xhtml/vocab#",
            dc: "http://purl.org/dc/elements/1.1/",
            foaf: "http://xmlthis.ns.com/foaf/0.1/",
            cc: "http://creativecommothis.ns.org/ns#",
            "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
            "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
            "xsd": "http://www.w3.org/TR/xmlschema-2/#"
        },
        reserved: ['alternate', 'appendix', 'bookmark', 'cite', 'chapter', 'contents', 'copyright',
            'first', 'glossary', 'help', 'icon', 'index', 'last', 'license', 'meta', 'next',
            'p3pv1', 'prev', 'role', 'section', 'stylesheet', 'subsection', 'start', 'top', 'up'],
        reservedNamespace: "http://www.w3.org/1999/xhtml/vocab#",
        defaultNamespace: undefined,
        charcase: 'preserve'
    };

    /**
     * Creates a {@link jQuery.uri} object by parsing a safe CURIE string (a CURIE
     * contained within square brackets). If the input safeCurie string does not
     * start with '[' and end with ']', the entire string content will be interpreted
     * as a URI string.
     * @methodOf jQuery
     * @param {String} safeCurie The safe CURIE string to be parsed.
     * @param {Object} [options] CURIE parsing options
     * @param {string} [options.reservedNamespace='http://www.w3.org/1999/xhtml/vocab#'] The namespace to
apply to a CURIE that has no prefix and either starts with a colon or is in the list of reserved local
names
     * @param {string} [options.defaultNamespace]  The namespace to apply to a CURIE with no prefix which
is not mapped to the reserved namespace by the rules given above.
     * @param {Object} [options.namespaces] A map of namespace bindings used to map CURIE prefixes to
URIs.
     * @param {string[]} [options.reserved=['alternate', 'appendix', 'bookmark', 'cite', 'chapter',
'contents', 'copyright',
     'first', 'glossary', 'help', 'icon', 'index', 'last', 'license', 'meta', 'next',
     'p3pv1', 'prev', 'role', 'section', 'stylesheet', 'subsection', 'start', 'top', 'up']]
     A list of local names that will always be mapped to the URI specified by reservedNamespace.
     * @param {string} [options.charcase='lower'] Specifies whether the curie's case is altered before
it's interpreted. Acceptable values are:
     * <dl>
     * <dt>lower</dt><dd>Force the CURIE string to lower case.</dd>
     * <dt>upper</dt><dd>Force the CURIE string to upper case.</dd>
     * <dt>preserve</dt><dd>Preserve the original case of the CURIE. Note that this might not be possible
if the CURIE has been taken from an HTML attribute value because of the case conversions performed
automatically by browsers. For this reason, it's a good idea to avoid mixed-case CURIEs within RDFa.</dd>
     * </dl>
     * @returns {jQuery.uri} A new {@link jQuery.uri} object representing the full absolute URI specified
by the CURIE.
     */
    Curie.safeCurie = function (safeCurie, options) {
        var m = /^\[([^\]]+)\]$/.exec(safeCurie);
        return m ? Curie(m[1], options) : URI(safeCurie);
    };

    /**
     * Creates a CURIE string from a URI string.
     * @methodOf jQuery
```

```
     * @param {String} uri The URI string to be converted to a CURIE.
     * @param {Object} [options] CURIE parsing options
     * @param {string} [options.reservedNamespace='http://www.w3.org/1999/xhtml/vocab#']
     *        If the input URI starts with this value, the generated CURIE will
     *        have no namespace prefix and will start with a colon character (:),
     *        unless the local part of the CURIE is one of the reserved names specified
     *        by the reservedNames option (see below), in which case the generated
     *        CURIE will have no namespace prefix and will not start with a colon
     *        character.
     * @param {string} [options.defaultNamespace]  If the input URI starts with this value, the generated
CURIE will have no namespace prefix and will not start with a colon.
     * @param {Object} [options.namespaces] A map of namespace bindings used to map CURIE prefixes to
URIs.
     * @param {string[]} [options.reserved=['alternate', 'appendix', 'bookmark', 'cite', 'chapter',
'contents', 'copyright',
     'first', 'glossary', 'help', 'icon', 'index', 'last', 'license', 'meta', 'next',
     'p3pv1', 'prev', 'role', 'section', 'stylesheet', 'subsection', 'start', 'top', 'up']]
     A list of local names that will always be mapped to the URI specified by reservedNamespace.
     * @param {string} [options.charcase='lower'] Specifies the case normalisation done to the CURIE.
Acceptable values are:
     * <dl>
     * <dt>lower</dt><dd>Normalise the CURIE to lower case.</dd>
     * <dt>upper</dt><dd>Normalise the CURIE to upper case.</dd>
     * <dt>preserve</dt><dd>Preserve the original case of the CURIE. Note that this might not be possible
if the CURIE has been taken from an HTML attribute value because of the case conversions performed
automatically by browsers. For this reason, it's a good idea to avoid mixed-case CURIEs within RDFa.</dd>
     * </dl>
     * @returns {jQuery.uri} A new {@link jQuery.uri} object representing the full absolute URI specified
by the CURIE.
     */
    Curie.createCurie = function (uri, options) {
        var opts = Utils.extend({}, Curie.defaults, options || {}),
            ns = opts.namespaces,
            curie   ;
        uri = URI(uri).toString();
        if (opts.reservedNamespace !== undefined && uri.substring(0, opts.reservedNamespace.toString
().length) === opts.reservedNamespace.toString()) {
            curie = uri.substring(opts.reservedNamespace.toString().length);
            if (Utils.indexOf(opts.reserved, curie) === -1) {
                curie = ':' + curie;
            }
        } else {
            Utils.each(ns, function (namespace, prefix) {
                if (uri.substring(0, namespace.toString().length) === namespace.toString()) {
                    curie = prefix + ':' + uri.substring(namespace.toString().length);
                    return null;
                }
            });
        }
        if (curie === undefined) {
            throw "No Namespace Binding: There's no appropriate namespace binding for generating a CURIE
from " + uri;
        } else {
            return curie;
        }
    };
    return Curie;
});define([
    "./uri"
], function (URI) {
    /*
     * jQuery CURIE @VERSION
     *
     * Copyright (c) 2008,2009 Jeni Tennison
     * Licensed under the MIT (MIT-LICENSE.txt)
     *
     * Depends:
```

```
 *  jquery.uri.js
 */
/**
 * @fileOverview XML Schema datatype handling
 * @author <a href="mailto:jeni@jenitennison.com">Jeni Tennison</a>
 * @copyright (c) 2008,2009 Jeni Tennison
 * @license MIT license (MIT-LICENSE.txt)
 * @version 1.0
 * @requires jquery.uri.js
 */

var strip = function (value) {
    return value.replace(/[ \t\n\r]+/, ' ').replace(/^ +/, '').replace(/ +$/, '');
};

/**
 * Creates a new jQuery.typedValue object. This should be invoked as a method
 * rather than constructed using new.
 * @class Represents a value with an XML Schema datatype
 * @param {String} value The string representation of the value
 * @param {String} datatype The XML Schema datatype URI
 * @returns {jQuery.typedValue}
 * @example intValue = jQuery.typedValue('42', 'http://www.w3.org/2001/XMLSchema#integer');
 */
typedValue = function (value, datatype) {
    return typedValue.fn.init(value, datatype);
};

typedValue.fn = typedValue.prototype = {
    /**
     * The string representation of the value
     * @memberOf jQuery.typedValue#
     */
    representation: undefined,
    /**
     * The value as an object. The type of the object will
     * depend on the XML Schema datatype URI specified
     * in the constructor. The following table lists the mappings
     * currently supported:
     * <table>
     *   <tr>
     *   <th>XML Schema Datatype</th>
     *   <th>Value type</th>
     *   </tr>
     *   <tr>
     *     <td>http://www.w3.org/2001/XMLSchema#string</td>
     *     <td>string</td>
     *   </tr>
     *   <tr>
     *     <td>http://www.w3.org/2001/XMLSchema#token</td>
     *     <td>string</td>
     *   </tr>
     *   <tr>
     *     <td>http://www.w3.org/2001/XMLSchema#NCName</td>
     *     <td>string</td>
     *   </tr>
     *   <tr>
     *     <td>http://www.w3.org/2001/XMLSchema#boolean</td>
     *     <td>bool</td>
     *   </tr>
     *   <tr>
     *     <td>http://www.w3.org/2001/XMLSchema#decimal</td>
     *     <td>string</td>
     *   </tr>
     *   <tr>
     *     <td>http://www.w3.org/2001/XMLSchema#integer</td>
     *     <td>int</td>
```

```
         *   </tr>
         *   <tr>
         *     <td>http://www.w3.org/2001/XMLSchema#int</td>
         *     <td>int</td>
         *   </tr>
         *   <tr>
         *     <td>http://www.w3.org/2001/XMLSchema#float</td>
         *     <td>float</td>
         *   </tr>
         *   <tr>
         *     <td>http://www.w3.org/2001/XMLSchema#double</td>
         *     <td>float</td>
         *   </tr>
         *   <tr>
         *     <td>http://www.w3.org/2001/XMLSchema#dateTime</td>
         *     <td>string</td>
         *   </tr>
         *   <tr>
         *     <td>http://www.w3.org/2001/XMLSchema#date</td>
         *     <td>string</td>
         *   </tr>
         *   <tr>
         *     <td>http://www.w3.org/2001/XMLSchema#gYear</td>
         *     <td>int</td>
         *   </tr>
         *   <tr>
         *     <td>http://www.w3.org/2001/XMLSchema#gMonthDay</td>
         *     <td>string</td>
         *   </tr>
         *   <tr>
         *     <td>http://www.w3.org/2001/XMLSchema#anyURI</td>
         *     <td>{@link jQuery.uri}</td>
         *   </tr>
         * </table>
         * @memberOf jQuery.typedValue#
         */
        value: undefined,
        /**
         * The XML Schema datatype URI for the value's datatype
         * @memberOf jQuery.typedValue#
         */
        datatype: undefined,

        init: function (value, datatype) {
            var d = typedValue.types[datatype];
            if (typedValue.valid(value, datatype)) {
                this.representation = value;
                this.datatype = datatype;
                this.value = d === undefined ? strip(value) : d.value(d.strip ? strip(value) : value);
                return this;
            } else {
                throw {
                    name: 'InvalidValue',
                    message: value + ' is not a valid ' + datatype + ' value'
                };
            }
        }
    };

    typedValue.fn.init.prototype = typedValue.fn;

    /**
     * An object that holds the datatypes supported by the script. The properties of this object are the
URIs of the datatypes, and each datatype has four properties:
     * <dl>
     *   <dt>strip</dt>
     *   <dd>A boolean value that indicates whether whitespace should be stripped from the value prior to
```

```
testing against the regular expression or passing to the value function.</dd>
     *   <dt>regex</dt>
     *   <dd>A regular expression that valid values of the type must match.</dd>
     *   <dt>validate</dt>
     *   <dd>Optional. A function that performs further testing on the value.</dd>
     *   <dt>value</dt>
     *   <dd>A function that returns a Javascript object equivalent for the value.</dd>
     * </dl>
     * You can add to this object as necessary for your own datatypes, and {@link jQuery.typedValue} and
{@link jQuery.typedValue.valid} will work with them.
     * @see jQuery.typedValue
     * @see jQuery.typedValue.valid
     */
    typedValue.types = {};

    typedValue.types['http://www.w3.org/2001/XMLSchema#string'] = {
        regex: /^.*$/,
        strip: false,
        /** @ignore */
        value: function (v) {
            return v;
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#token'] = {
        regex: /^.*$/,
        strip: true,
        /** @ignore */
        value: function (v) {
            return strip(v);
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#NCName'] = {
        regex: /^[a-z_][-\.a-z0-9]+$/i,
        strip: true,
        /** @ignore */
        value: function (v) {
            return strip(v);
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#boolean'] = {
        regex: /^(?:true|false|1|0)$/,
        strip: true,
        /** @ignore */
        value: function (v) {
            return v === 'true' || v === '1';
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#decimal'] = {
        regex: /^[\-\+]?(?:[0-9]+\.[0-9]*|\.[0-9]+|[0-9]+)$/,
        strip: true,
        /** @ignore */
        value: function (v) {
            v = v.replace(/^0+/, '')
                .replace(/0+$/, '');
            if (v === '') {
                v = '0.0';
            }
            if (v.substring(0, 1) === '.') {
                v = '0' + v;
            }
            if (/\.$/.test(v)) {
                v = v + '0';
            } else if (!/\./.test(v)) {
```

```
                v = v + '.0';
            }
            return v;
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#integer'] = {
        regex: /^[\-\+]?[0-9]+$/,
        strip: true,
        /** @ignore */
        value: function (v) {
            return parseInt(v, 10);
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#int'] = {
        regex: /^[\-\+]?[0-9]+$/,
        strip: true,
        /** @ignore */
        value: function (v) {
            return parseInt(v, 10);
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#float'] = {
        regex: /^(?:[\-\+]?(?:[0-9]+\.[0-9]*|\.[0-9]+|[0-9]+)(?:[eE][\-\+]?[0-9]+)?|[\-\+]?INF|NaN)$/,
        strip: true,
        /** @ignore */
        value: function (v) {
            if (v === '-INF') {
                return -1 / 0;
            } else if (v === 'INF' || v === '+INF') {
                return 1 / 0;
            } else {
                return parseFloat(v);
            }
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#double'] = {
        regex: typedValue.types['http://www.w3.org/2001/XMLSchema#float'].regex,
        strip: true,
        value: typedValue.types['http://www.w3.org/2001/XMLSchema#float'].value
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#duration'] = {
        regex: /^([\-\+])?P(?:([0-9]+)Y)?(?:([0-9]+)M)?(?:([0-9]+)D)?(?:T(?:([0-9]+)H)?(?:([0-9]+)M)?(?:
([0-9]+(?:\.[0-9]+)?)?S)?)$/,
        /** @ignore */
        validate: function (v) {
            var m = this.regex.exec(v);
            return m[2] || m[3] || m[4] || m[5] || m[6] || m[7];
        },
        strip: true,
        /** @ignore */
        value: function (v) {
            return v;
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#yearMonthDuration'] = {
        regex: /^([\-\+])?P(?:([0-9]+)Y)?(?:([0-9]+)M)?$/,
        /** @ignore */
        validate: function (v) {
            var m = this.regex.exec(v);
            return m[2] || m[3];
        },
```

```
        strip: true,
        /** @ignore */
        value: function (v) {
            var m = this.regex.exec(v),
                years = m[2] || 0,
                months = m[3] || 0;
            months += years * 12;
            return m[1] === '-' ? -1 * months : months;
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#dateTime'] = {
        regex: /^(-?[0-9]{4,})-([0-9]{2})-([0-9]{2})T([0-9]{2}):([0-9]{2}):(([0-9]{2})(\.([0-9]+))?)((?:
[\-\+]([0-9]{2}):([0-9]{2}))|Z)?$/,
        /** @ignore */
        validate: function (v) {
            var
                m = this.regex.exec(v),
                year = parseInt(m[1], 10),
                tz = m[10] === undefined || m[10] === 'Z' ? '+0000' : m[10].replace(/:/, ''),
                date;
            if (year === 0 ||
                parseInt(tz, 10) < -1400 || parseInt(tz, 10) > 1400) {
                return false;
            }
            try {
                year = year < 100 ? Math.abs(year) + 1000 : year;
                month = parseInt(m[2], 10);
                day = parseInt(m[3], 10);
                if (day > 31) {
                    return false;
                } else if (day > 30 && !(month === 1 || month === 3 || month === 5 || month === 7 ||
month === 8 || month === 10 || month === 12)) {
                    return false;
                } else if (month === 2) {
                    if (day > 29) {
                        return false;
                    } else if (day === 29 && (year % 4 !== 0 || (year % 100 === 0 && year % 400 !== 0))) {
                        return false;
                    }
                }
                date = '' + year + '/' + m[2] + '/' + m[3] + ' ' + m[4] + ':' + m[5] + ':' + m[7] + ' ' +
tz;
                date = new Date(date);
                return true;
            } catch (e) {
                return false;
            }
        },
        strip: true,
        /** @ignore */
        value: function (v) {
            return v;
        }
    };

    typedValue.types['http://www.w3.org/2001/XMLSchema#date'] = {
        regex: /^(-?[0-9]{4,})-([0-9]{2})-([0-9]{2})((?:[\-\+]([0-9]{2}):([0-9]{2}))|Z)?$/,
        /** @ignore */
        validate: function (v) {
            var
                m = this.regex.exec(v),
                year = parseInt(m[1], 10),
                month = parseInt(m[2], 10),
                day = parseInt(m[3], 10),
                tz = m[10] === undefined || m[10] === 'Z' ? '+0000' : m[10].replace(/:/, '');
            if (year === 0 ||
```

```
            month > 12 ||
            day > 31 ||
            parseInt(tz, 10) < -1400 || parseInt(tz, 10) > 1400) {
            return false;
        } else {
            return true;
        }
    },
    strip: true,
    /** @ignore */
    value: function (v) {
        return v;
    }
};

typedValue.types['http://www.w3.org/2001/XMLSchema#gYear'] = {
    regex: /^-?([0-9]{4,})$/,
    /** @ignore */
    validate: function (v) {
        var i = parseInt(v, 10);
        return i !== 0;
    },
    strip: true,
    /** @ignore */
    value: function (v) {
        return parseInt(v, 10);
    }
};

typedValue.types['http://www.w3.org/2001/XMLSchema#gMonthDay'] = {
    regex: /^--([0-9]{2})-([0-9]{2})((?:[\-\+]([0-9]{2}):([0-9]{2}))|Z)?$/,
    /** @ignore */
    validate: function (v) {
        var
            m = this.regex.exec(v),
            month = parseInt(m[1], 10),
            day = parseInt(m[2], 10),
            tz = m[3] === undefined || m[3] === 'Z' ? '+0000' : m[3].replace(/:/, '');
        if (month > 12 ||
            day > 31 ||
            parseInt(tz, 10) < -1400 || parseInt(tz, 10) > 1400) {
            return false;
        } else if (month === 2 && day > 29) {
            return false;
        } else if ((month === 4 || month === 6 || month === 9 || month === 11) && day > 30) {
            return false;
        } else {
            return true;
        }
    },
    strip: true,
    /** @ignore */
    value: function (v) {
        return v;
    }
};

typedValue.types['http://www.w3.org/2001/XMLSchema#anyURI'] = {
    regex: /^.*$/,
    strip: true,
    /** @ignore */
    value: function (v, options) {
        var opts = graphite.extend({}, typedValue.defaults, options);
        return URI.resolve(v, opts.base);
    }
};
```

```
    typedValue.defaults = {
        namespaces: {}
    };

    /**
     * Checks whether a value is valid according to a given datatype. The datatype must be held in the
{@link jQuery.typedValue.types} object.
     * @param {String} value The value to validate.
     * @param {String} datatype The URI for the datatype against which the value will be validated.
     * @returns {boolean} True if the value is valid or the datatype is not recognised.
     * @example validDate = $.typedValue.valid(date, 'http://www.w3.org/2001/XMLSchema#date');
     */
    typedValue.valid = function (value, datatype) {
        var d = typedValue.types[datatype];
        if (d === undefined) {
            return true;
        } else {
            value = d.strip ? strip(value) : value;
            if (d.regex.test(value)) {
                return d.validate === undefined ? true : d.validate(value);
            } else {
                return false;
            }
        }
    };
    return typedValue;
});define([
    "./curie",
    "./datatype",
    "./uri",
    "./../graphite/utils"
], function (CURIE, TypedValue, URI, Utils) {
    /*
     * jQuery RDF @VERSION
     *
     * Copyright (c) 2008,2009 Jeni Tennison
     * Licensed under the MIT (MIT-LICENSE.txt)
     *
     * Depends:
     *  jquery.uri.js
     *  jquery.xmlns.js
     *  jquery.datatype.js
     *  jquery.curie.js
     *  jquery.json.js
     */
    /**
     * @fileOverview jQuery RDF
     * @author <a href="mailto:jeni@jenitennison.com">Jeni Tennison</a>
     * @copyright (c) 2008,2009 Jeni Tennison
     * @license MIT license (MIT-LICENSE.txt)
     * @version 1.0
     */
    /**
     * @exports $ as jQuery
     */
    var
        memResource = {},
        memBlank = {},
        memLiteral = {},
        memTriple = {},
        memPattern = {},

        xsdNs = "http://www.w3.org/2001/XMLSchema#",
        rdfNs = "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
        rdfsNs = "http://www.w3.org/2000/01/rdf-schema#",

        uriRegex = /^<(([^>]|\\>)*)>$/,
```

```
    literalRegex = /^("""((\\"|[^"])*)"""|"((\\"|[^"])*)")(@([a-z]+(-[a-z0-9]+)*)|\^\^(.+))?$/,
    tripleRegex = /(("""((\\"|[^"])*)""")|("(\\"|[^"]|)*")|(<(\\>|[^>])*>)|\S)+/g,

    blankNodeSeed = databankSeed = new Date().getTime() % 1000,
    blankNodeID = function () {
        blankNodeSeed += 1;
        return 'b' + blankNodeSeed.toString(16);
    },

    databankID = function () {
        databankSeed += 1;
        return 'data' + databankSeed.toString(16);
    },
    databanks = {},

    documentQueue = {},

    subject = function (subject, opts) {
        if (typeof subject === 'string') {
            try {
                return rdf.resource(subject, opts);
            } catch (e) {
                try {
                    return rdf.blank(subject, opts);
                } catch (f) {
                    throw "Bad Triple: Subject " + subject + " is not a resource: " + f;
                }
            }
        } else {
            return subject;
        }
    },

    property = function (property, opts) {
        if (property === 'a') {
            return rdf.type;
        } else if (typeof property === 'string') {
            try {
                return rdf.resource(property, opts);
            } catch (e) {
                throw "Bad Triple: Property " + property + " is not a resource: " + e;
            }
        } else {
            return property;
        }
    },

    object = function (object, opts) {
        if (typeof object === 'string') {
            try {
                return rdf.resource(object, opts);
            } catch (e) {
                try {
                    return rdf.blank(object, opts);
                } catch (f) {
                    try {
                        return rdf.literal(object, opts);
                    } catch (g) {
                        throw "Bad Triple: Object " + object + " is not a resource or a literal " + g;
                    }
                }
            }
        } else {
            return object;
        }
    },
```

```
        testResource = function (resource, filter, existing) {
            var variable;
            if (typeof filter === 'string') {
                variable = filter.substring(1);
                if (existing[variable] && existing[variable] !== resource) {
                    return null;
                } else {
                    existing[variable] = resource;
                    return existing;
                }
            } else if (filter === resource) {
                return existing;
            } else {
                return null;
            }
        },

        findMatches = function (databank, pattern) {
            if (databank.union === undefined) {
                if (pattern.subject.type !== undefined) {
                    if (databank.subjectIndex[pattern.subject] === undefined) {
                        return [];
                    }
                    return Utils.map(databank.subjectIndex[pattern.subject], function (triple) {
                        var bindings = pattern.exec(triple);
                        return bindings === null ? null : { bindings: bindings, triples: [triple] };
                    });
                } else if (pattern.object.type === 'uri' || pattern.object.type === 'bnode') {
                    if (databank.objectIndex[pattern.object] === undefined) {
                        return [];
                    }
                    return Utils.map(databank.objectIndex[pattern.object], function (triple) {
                        var bindings = pattern.exec(triple);
                        return bindings === null ? null : { bindings: bindings, triples: [triple] };
                    });
                } else if (pattern.property.type !== undefined) {
                    if (databank.propertyIndex[pattern.property] === undefined) {
                        return [];
                    }
                    return Utils.without(Utils.map(databank.propertyIndex[pattern.property], function
(triple) {
                        var bindings = pattern.exec(triple);
                        return bindings === null ? null : { bindings: bindings, triples: [triple] };
                    }), null);
                }
            }
            return Utils.without(Utils.flatten(Utils.map(databank.triples(), function (triple) {
                var bindings = pattern.exec(triple);
                return bindings === null ? null : { bindings: bindings, triples: [triple] };
            })), null);
        },

        mergeMatches = function (existingMs, newMs, optional) {
            var compatibleMs,
                matches = Utils.without(Utils.flatten(Utils.map(existingMs, function (existingM) {
                    compatibleMs = Utils.flatten(Utils.map(newMs, function (newM) {
                        // For newM to be compatible with existingM, all the bindings
                        // in newM must either be the same as in existingM, or not
                        // exist in existingM
                        var k, b, isCompatible = true;
                        for (k in newM.bindings) {
                            b = newM.bindings[k];
                            if (!(existingM.bindings[k] === undefined ||
                                existingM.bindings[k] === b)) {
                                isCompatible = false;
                                break;
                            }
```

```
                }
                return isCompatible ? newM : null;
            }));
            compatibleMs = Utils.without(compatibleMs, null);
            if (compatibleMs.length > 0) {
                return Utils.flatten(Utils.map(compatibleMs, function (compatibleM) {
                    return {
                        bindings: Utils.extend({}, existingM.bindings, compatibleM.bindings),
                        triples: unique(existingM.triples.concat(compatibleM.triples))
                    };
                }));
            } else {
                return optional ? existingM : null;
            }
        })), null);
    return matches;
},

registerQuery = function (databank, query) {
    var s, p, o;
    if (query.filterExp !== undefined && !Utils.isFunction(query.filterExp)) {
        if (databank.union === undefined) {
            s = typeof query.filterExp.subject === 'string' ? '' : query.filterExp.subject;
            p = typeof query.filterExp.property === 'string' ? '' : query.filterExp.property;
            o = typeof query.filterExp.object === 'string' ? '' : query.filterExp.object;
            if (databank.queries[s] === undefined) {
                databank.queries[s] = {};
            }
            if (databank.queries[s][p] === undefined) {
                databank.queries[s][p] = {};
            }
            if (databank.queries[s][p][o] === undefined) {
                databank.queries[s][p][o] = [];
            }
            databank.queries[s][p][o].push(query);
        } else {
            Utils.each(databank.union, function (databank) {
                registerQuery(databank, query);
            });
        }
    }
},

resetQuery = function (query) {
    query.length = 0;
    query.matches = [];
    Utils.each(query.children, function (child) {
        resetQuery(child);
    });
    Utils.each(query.partOf, function (union) {
        resetQuery(union);
    });
},

updateQuery = function (query, matches) {
    if (matches.length > 0) {
        Utils.each(query.children, function (child, i) {
            leftActivate(child, matches);
        });
        Utils.each(query.partOf, function (union, i) {
            updateQuery(union, matches);
        });
        Utils.each(matches, function (match, i) {
            if(match) {
                query.matches.push(match);
                Array.prototype.push.call(query, match.bindings);
            }
```

```
            });
        }
    },

    filterMatches = function (matches, variables) {
        var bindings,
            triples,
            value,
            nvariables = variables.length,
            match = {},
            keyobject = {},
            keys = {},
            filtered = [];
        Utils.each(matches, function (m) {
            bindings = m.bindings;
            triples = m.triples;
            keyobject = keys;
            Utils.each(variables, function (variable, j) {
                value = bindings[variable];
                if (j === nvariables - 1) {
                    if (keyobject[value] === undefined) {
                        match = { bindings: {}, triples: triples };
                        Utils.each(variables, function (v) {
                            match.bindings[v] = bindings[v];
                        });
                        keyobject[value] = match;
                        filtered.push(match);
                    } else {
                        match = keyobject[value];
                        match.triples = match.triples.concat(triples);
                    }
                } else {
                    if (keyobject[value] === undefined) {
                        keyobject[value] = {};
                    }
                    keyobject = keyobject[value];
                }
            });
        })
        return filtered;
    },

    renameMatches = function (matches, old) {
        var newMatch,
            keys = {},
            renamed = [],
            i = 0;
        if(Utils.isArray(matches[0])) {
            Utils.each(matches, function (match) {
                renamed.push(renameMatches(match, old));
            });
        } else {
            Utils.each(matches, function (match, key) {
                if (keys[match.bindings[old]] === undefined) {
                    newMatch = {
                        bindings: { node: match.bindings[old] },
                        triples: match.triples
                    };
                    renamed.push(newMatch);
                    keys[match.bindings[old]] = newMatch;
                } else {
                    newMatch = keys[match.bindings[old]];
                    newMatch.triples = newMatch.triples.concat(match.triples);
                }
            });
        }
        return renamed;
```

```
        },

        leftActivate = function (query, matches) {
            var newMatches;
            if (query.union === undefined) {
                if (query.top || query.parent.top) {
                    newMatches = query.alphaMemory;
                } else {
                    matches = matches || query.parent.matches;
                    if (Utils.isFunction(query.filterExp)) {
                        newMatches = Utils.flatten(Utils.map(matches, function (match, i) {
                            return query.filterExp.call(match.bindings, i, match.bindings,
match.triples) ? match : null;
                        }));
                    } else if (query.filterExp !== undefined) {
                        newMatches = mergeMatches(matches, query.alphaMemory, query.filterExp.optional);
                    } else {
                        newMatches = matches;
                    }
                }
            } else {
                newMatches = Utils.flatten(Utils.map(query.union, function (q) {
                    return q.matches;
                }));
            }
            if (query.selections !== undefined) {
                newMatches = filterMatches(newMatches, query.selections);
            } else if (query.navigate !== undefined) {
                newMatches = renameMatches(newMatches, query.navigate);
            }
            updateQuery(query, newMatches);
        },

        rightActivate = function (query, match) {
            var newMatches;
            if (query.filterExp.optional) {
                resetQuery(query);
                leftActivate(query);
            } else {
                if (query.top || query.parent.top) {
                    newMatches = [match];
                } else {
                    newMatches = mergeMatches(query.parent.matches, [match], false);
                }
                updateQuery(query, Utils.without(newMatches, null));
            }
        },

        addToQuery = function (query, triple) {
            var match,
                bindings = query.filterExp.exec(triple);
            if (bindings !== null) {
                match = { triples: [triple], bindings: bindings };
                query.alphaMemory.push(match);
                rightActivate(query, match);
            }
        },

        removeFromQuery = function (query, triple) {
            query.alphaMemory.splice(Utils.indexOf(query.alphaMindexOfemory, triple), 1);
            resetQuery(query);
            leftActivate(query);
        },

        addToQueries = function (queries, triple) {
            Utils.each(queries, function (query, i) {
                addToQuery(query, triple);
```

```
        });
    },

    removeFromQueries = function (queries, triple) {
        Utils.each(queries, function (query, i) {
            removeFromQuery(query, triple);
        });
    },

    addToDatabankQueries = function (databank, triple) {
        var s = triple.subject,
            p = triple.property,
            o = triple.object;
        if (databank.union === undefined) {
            if (databank.queries[s] !== undefined) {
                if (databank.queries[s][p] !== undefined) {
                    if (databank.queries[s][p][o] !== undefined) {
                        addToQueries(databank.queries[s][p][o], triple);
                    }
                    if (databank.queries[s][p][''] !== undefined) {
                        addToQueries(databank.queries[s][p][''], triple);
                    }
                }
                if (databank.queries[s][''] !== undefined) {
                    if (databank.queries[s][''][o] !== undefined) {
                        addToQueries(databank.queries[s][''][o], triple);
                    }
                    if (databank.queries[s][''][''] !== undefined) {
                        addToQueries(databank.queries[s][''][''], triple);
                    }
                }
            }
            if (databank.queries[''] !== undefined) {
                if (databank.queries[''][p] !== undefined) {
                    if (databank.queries[''][p][o] !== undefined) {
                        addToQueries(databank.queries[''][p][o], triple);
                    }
                    if (databank.queries[''][p][''] !== undefined) {
                        addToQueries(databank.queries[''][p][''], triple);
                    }
                }
                if (databank.queries[''][''] !== undefined) {
                    if (databank.queries[''][''][o] !== undefined) {
                        addToQueries(databank.queries[''][''][o], triple);
                    }
                    if (databank.queries[''][''][''] !== undefined) {
                        addToQueries(databank.queries[''][''][''], triple);
                    }
                }
            }
        } else {
            Utils.each(databank.union, function (databank, i) {
                addToDatabankQueries(databank, triple);
            });
        }
    },

    removeFromDatabankQueries = function (databank, triple) {
        var s = triple.subject,
            p = triple.property,
            o = triple.object;
        if (databank.union === undefined) {
            if (databank.queries[s] !== undefined) {
                if (databank.queries[s][p] !== undefined) {
                    if (databank.queries[s][p][o] !== undefined) {
                        removeFromQueries(databank.queries[s][p][o], triple);
                    }
```

```
                    if (databank.queries[s][p][''] !== undefined) {
                        removeFromQueries(databank.queries[s][p][''], triple);
                    }
                }
                if (databank.queries[s][''] !== undefined) {
                    if (databank.queries[s][''][o] !== undefined) {
                        removeFromQueries(databank.queries[s][''][o], triple);
                    }
                    if (databank.queries[s][''][''] !== undefined) {
                        removeFromQueries(databank.queries[s][''][''], triple);
                    }
                }
            }
            if (databank.queries[''] !== undefined) {
                if (databank.queries[''][p] !== undefined) {
                    if (databank.queries[''][p][o] !== undefined) {
                        removeFromQueries(databank.queries[''][p][o], triple);
                    }
                    if (databank.queries[''][p][''] !== undefined) {
                        removeFromQueries(databank.queries[''][p][''], triple);
                    }
                }
                if (databank.queries[''][''] !== undefined) {
                    if (databank.queries[''][''][o] !== undefined) {
                        removeFromQueries(databank.queries[''][''][o], triple);
                    }
                    if (databank.queries[''][''][''] !== undefined) {
                        removeFromQueries(databank.queries[''][''][''], triple);
                    }
                }
            }
        } else {
            Utils.each(databank.union, function (databank, i) {
                removeFromDatabankQueries(databank, triple);
            });
        }
    },

    group = function (bindings, variables, base) {
        var variable = variables[0], grouped = {}, results = [], i, newbase;
        base = base || {};
        if (variables.length === 0) {
            for (i = 0; i < bindings.length; i += 1) {
                for (v in bindings[i]) {
                    if (base[v] === undefined) {
                        base[v] = [];
                    }
                    if (Utils.isArray(base[v])) {
                        base[v].push(bindings[i][v]);
                    }
                }
            }
            return [base];
        }
        // collect together the grouped results
        for (i = 0; i < bindings.length; i += 1) {
            key = bindings[i][variable];
            if (grouped[key] === undefined) {
                grouped[key] = [];
            }
            grouped[key].push(bindings[i]);
        }
        // call recursively on each group
        variables = variables.splice(1, 1);
        for (v in grouped) {
            newbase = Utils.extend({}, base);
            newbase[variable] = grouped[v][0][variable];
```

```
                results = results.concat(group(grouped[v], variables, newbase));
            }
            return results;
        },

        queue = function (databank, url, callbacks) {
            if (documentQueue[databank.id] === undefined) {
                documentQueue[databank.id] = {};
            }
            if (documentQueue[databank.id][url] === undefined) {
                documentQueue[databank.id][url] = callbacks;
                return false;
            }
            return true;
        },

        dequeue = function (databank, url, result, args) {
            var callbacks = documentQueue[databank.id][url];
            if (Utils.isFunction(callbacks[result])) {
                callbacks[result].call(databank, args);
            }
            documentQueue[databank.id][url] = undefined;
        },

        unique = function( b ) {
            var a = [];
            var l = b.length;
            for(var i=0; i<l; i++) {
                for(var j=i+1; j<l; j++) {
                    // If b[i] is found later in the array
                    if (b[i] === b[j])
                        j = ++i;
                }
                a.push(b[i]);
            }
            return a;
        };


    TypedValue.types['http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral'] = {
        regex: /^.*$/m,
        strip: false,
        value: function (v) {
            return v;
        }
    };

    /**
     * <p>Creates a new jQuery.rdf object. This should be invoked as a method rather than constructed
using new; indeed you will usually want to generate these objects using a method such as {@link
jQuery#rdf} or {@link jQuery.rdf#where}.</p>
     * @class <p>A jQuery.rdf object represents the results of a query over its {@link
jQuery.rdf#databank}. The results of a query are a sequence of objects which represent the bindings of
values to the variables used in filter expressions specified using {@link jQuery.rdf#where} or {@link
jQuery.rdf#optional}. Each of the objects in this sequence has associated with it a set of triples that
are the sources for the variable bindings, which you can get at using {@link jQuery.rdf#sources}.</p>
     * <p>The {@link jQuery.rdf} object itself is a lot like a {@link jQuery} object. It has a {@link
jQuery.rdf#length} and the individual matches can be accessed using <code>[<var>n</var>]</code>, but you
can also iterate through the matches using {@link jQuery.rdf#map} or {@link jQuery.rdf#each}.</p>
     * <p>{@link jQuery.rdf} is designed to mirror the functionality of <a href="http://www.w3.org/TR/rdf-
sparql-query/">SPARQL</a> while providing an interface that's familiar and easy to use for jQuery
programmers.</p>
     * @param {Object} [options]
     * @param {jQuery.rdf.databank} [options.databank] The databank that this query should operate over.
     * @param {jQuery.rdf.triple[]} [options.triples] A set of triples over which the query operates;
this is only used if options.databank isn't specified, in which case a new databank with these triples is
generated.
```

```
     * @param {Object} [options.namespaces] An object representing a set of namespace bindings. Rather
than passing this in when you construct the {@link jQuery.rdf} instance, you will usually want to use the
{@link jQuery.rdf#prefix} method.
     * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs used
within the query.
     * @returns {Object}
     * @example rdf = jQuery.rdf();
     * @see jQuery#rdf
     */
    rdf = function (options) {
        return new rdf.fn.init(options);
    };

    rdf.fn = rdf.prototype = {
        /**
         * The version of rdfQuery.
         * @type String
         */
        rdfquery: '1.1',

        init: function (options) {
            var databanks, i;
            options = options || {};
            /* must specify either a parent or a union, otherwise it's the top */
            this.parent = options.parent;
            this.union = options.union;
            this.top = this.parent === undefined && this.union === undefined;
            if (this.union === undefined) {
                if (options.databank === undefined) {
                    /**
                     * The databank over which this query operates.
                     * @type jQuery.rdf.databank
                     */
                    this.databank = this.parent === undefined ? rdf.databank(options.triples, options) :
this.parent.databank;
                } else {
                    this.databank = options.databank;
                }
            } else {
                databanks = Utils.map(this.union, function (query) {
                    return query.databank;
                });
                databanks = unique(databanks);
                if (databanks[1] !== undefined) {
                    this.databank = rdf.databank(undefined, { union: databanks });
                } else {
                    this.databank = databanks[0];
                }
            }
            this.children = [];
            this.partOf = [];
            this.filterExp = options.filter;
            this.selections = options.distinct;
            this.navigate = options.navigate;
            this.alphaMemory = [];
            this.matches = [];
            /**
             * The number of matches represented by the {@link jQuery.rdf} object.
             * @type Integer
             */
            this.length = 0;
            if (this.filterExp !== undefined) {
                if (!Utils.isFunction(this.filterExp)) {
                    registerQuery(this.databank, this);
                    this.alphaMemory = findMatches(this.databank, this.filterExp);
                }
            } else if (options.nodes !== undefined) {
```

```
                this.alphaMemory = [];
                for (i = 0; i < options.nodes.length; i += 1) {
                    this.alphaMemory.push({
                        bindings: { node: options.nodes[i] },
                        triples: []
                    });
                }
            }
            leftActivate(this);
            return this;
        },

        /**
         * Sets or returns the base URI of the {@link jQuery.rdf#databank}.
         * @param {String|jQuery.uri} [base]
         * @returns A {@link jQuery.uri} if no base URI is specified, otherwise returns this {@link
jQuery.rdf} object.
         * @example baseURI = jQuery('html').rdf().base();
         * @example jQuery('html').rdf().base('http://www.example.org/');
         * @see jQuery.rdf.databank#base
         */
        base: function (base) {
            if (base === undefined) {
                return this.databank.base();
            } else {
                this.databank.base(base);
                return this;
            }
        },

        /**
         * Sets or returns a namespace binding on the {@link jQuery.rdf#databank}.
         * @param {String} [prefix]
         * @param {String} [namespace]
         * @returns {Object|jQuery.uri|jQuery.rdf} If no prefix or namespace is specified, returns an
object providing all namespace bindings on the {@link jQuery.rdf.databank}. If a prefix is specified
without a namespace, returns the {@link jQuery.uri} associated with that prefix. Otherwise returns this
{@link jQuery.rdf} object after setting the namespace binding.
         * @example namespace = jQuery('html').rdf().prefix('foaf');
         * @example jQuery('html').rdf().prefix('foaf', 'http://xmlns.com/foaf/0.1/');
         * @see jQuery.rdf.databank#prefix
         */
        prefix: function (prefix, namespace) {
            if (namespace === undefined) {
                return this.databank.prefix(prefix);
            } else {
                this.databank.prefix(prefix, namespace);
                return this;
            }
        },

        /**
         * Adds a triple to the {@link jQuery.rdf#databank} or another {@link jQuery.rdf} object to
create a union.
         * @param {String|jQuery.rdf.triple|jQuery.rdf.pattern|jQuery.rdf} triple The triple, {@link
jQuery.rdf.pattern} or {@link jQuery.rdf} object to be added to this one. If the triple is a {@link
jQuery.rdf} object, the two queries are unioned together. If the triple is a string, it's parsed as a
{@link jQuery.rdf.pattern}. The pattern will be completed using the current matches on the {@link
jQuery.rdf} object to create multiple triples, one for each set of bindings.
         * @param {Object} [options]
         * @param {Object} [options.namespaces] An object representing a set of namespace bindings used
to interpret CURIEs within the triple. Defaults to the namespace bindings defined on the {@link
jQuery.rdf#databank}.
         * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs
used within the triple. Defaults to the base URI defined on the {@link jQuery.rdf#databank}.
         * @returns {jQuery.rdf} This {@link jQuery.rdf} object.
         * @example
```

```
         * var rdf = rdf()
         *   .prefix('dc', ns.dc)
         *   .prefix('foaf', ns.foaf)
         *   .add('&lt;photo1.jpg> dc:creator &lt;http://www.blogger.com/profile/1109404> .')
         *   .add('&lt;http://www.blogger.com/profile/1109404> foaf:img &lt;photo1.jpg> .');
         * @example
         * var rdfA = rdf()
         *   .prefix('dc', ns.dc)
         *   .add('&lt;photo1.jpg> dc:creator "Jane"');
         * var rdfB = rdf()
         *   .prefix('foaf', ns.foaf)
         *   .add('&lt;photo1.jpg> foaf:depicts "Jane"');
         * var rdf = rdfA.add(rdfB);
         * @see jQuery.rdf.databank#add
         */
        add: function (triple, options) {
            var query, databank;
            if (triple.rdfquery !== undefined) {
                if (triple.top) {
                    databank = this.databank.add(triple.databank);
                    query = rdf({ parent: this.parent, databank: databank });
                    return query;
                } else if (this.top) {
                    databank = triple.databank.add(this.databank);
                    query = rdf({ parent: triple.parent, databank: databank });
                    return query;
                } else if (this.union === undefined) {
                    query = rdf({ union: [this, triple] });
                    this.partOf.push(query);
                    triple.partOf.push(query);
                    return query;
                } else {
                    this.union.push(triple);
                    triple.partOf.push(this);
                }
            } else {
                if (typeof triple === 'string') {
                    options = Utils.extend({}, { base: this.base(), namespaces: this.prefix(), source:
triple }, options);
                    triple = rdf.pattern(triple, options);
                }
                if (triple.isFixed()) {
                    this.databank.add(triple.triple(), options);
                } else {
                    query = this;
                    Utils.each(this, function (data) {
                        var t = triple.triple(data);
                        if (t !== null) {
                            query.databank.add(t, options);
                        }
                    });
                }
            }
            return this;
        },

        /**
         * Removes a triple or several triples from the {@link jQuery.rdf#databank}.
         * @param {String|jQuery.rdf.triple|jQuery.rdf.pattern} triple The triple to be removed, or a
{@link jQuery.rdf.pattern} that matches the triples that should be removed.
         * @param {Object} [options]
         * @param {Object} [options.namespaces] An object representing a set of namespace bindings used
to interpret any CURIEs within the triple or pattern. Defaults to the namespace bindings defined on the
{@link jQuery.rdf#databank}.
         * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs
used within the triple or pattern. Defaults to the base URI defined on the {@link jQuery.rdf#databank}.
         * @returns {jQuery.rdf} The {@link jQuery.rdf} object itself.
```

```
         * @example
         * var rdf = $('html').rdf()
         *   .prefix('foaf', ns.foaf)
         *   .where('?person foaf:givenname ?gname')
         *   .where('?person foaf:family_name ?fname')
         *   .remove('?person foaf:family_name ?fname');
         * @see jQuery.rdf.databank#remove
         */
        remove: function (triple, options) {
            var query;
            if (typeof triple === 'string') {
                options = Utils.extend({}, { base: this.base(), namespaces: this.prefix() }, options);
                triple = rdf.pattern(triple, options);
            }
            if (triple.isFixed()) {
                this.databank.remove(triple.triple(), options);
            } else {
                query = this;
                Utils.each(this, function (data) {
                    var t = triple.triple(data);
                    if (t !== null) {
                        query.databank.remove(t, options);
                    }
                });
            }
            return this;
        },

        /**
         * Loads some data into the {@link jQuery.rdf#databank}
         * @param data
         * @param {Object} [options]
         * @see jQuery.rdf.databank#load
         */
        load: function (data, options) {
            var rdf = this,
                options = options || {},
                success = options.success;
            if (success !== undefined) {
                options.success = function () {
                    success.call(rdf);
                }
            }
            this.databank.load(data, options);
            return this;
        },

        /**
         * Creates a new {@link jQuery.rdf} object whose databank contains all the triples in this
object's databank except for those in the argument's databank.
         * @param {jQuery.rdf} query
         * @see jQuery.rdf.databank#except
         */
        except: function (query) {
            return rdf({ databank: this.databank.except(query.databank) });
        },

        /**
         * Creates a new {@link jQuery.rdf} object that is the result of filtering the matches on this
{@link jQuery.rdf} object based on the filter that's passed into it.
         * @param {String|jQuery.rdf.pattern} filter An expression that filters the triples in the {@link
jQuery.rdf#databank} to locate matches based on the matches on this {@link jQuery.rdf} object. If it's a
string, the filter is parsed as a {@link jQuery.rdf.pattern}.
         * @param {Object} [options]
         * @param {Object} [options.namespaces] An object representing a set of namespace bindings used
to interpret any CURIEs within the pattern. Defaults to the namespace bindings defined on the {@link
jQuery.rdf#databank}.
```

```
         * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs
used within the pattern. Defaults to the base URI defined on the {@link jQuery.rdf#databank}.
         * @param {boolean} [options.optional] Not usually used (use {@link jQuery.rdf#optional} instead).
         * @returns {jQuery.rdf} A new {@link jQuery.rdf} object whose {@link jQuery.rdf#parent} is this
{@link jQuery.rdf}.
         * @see jQuery.rdf#optional
         * @see jQuery.rdf#filter
         * @see jQuery.rdf#about
         * @example
         * var rdf = rdf()
         *   .prefix('foaf', ns.foaf)
         *   .add('_:a foaf:givenname   "Alice" .')
         *   .add('_:a foaf:family_name "Hacker" .')
         *   .add('_:b foaf:givenname   "Bob" .')
         *   .add('_:b foaf:family_name "Hacker" .')
         *   .where('?person foaf:family_name "Hacker"')
         *   .where('?person foaf:givenname "Bob");
         */
        where: function (filter, options) {
            var query, base, namespaces, optional;
            options = options || {};
            if (typeof filter === 'string') {
                base = options.base || this.base();
                namespaces = Utils.extend({}, this.prefix(), options.namespaces || {});
                optional = options.optional || false;
                filter = rdf.pattern(filter, { namespaces: namespaces, base: base, optional: optional });
            }
            query = rdf(Utils.extend({}, options, { parent: this, filter: filter }));
            this.children.push(query);
            return query;
        },

        /**
         * Creates a new {@link jQuery.rdf} object whose set of bindings might optionally include those
based on the filter pattern.
         * @param {String|jQuery.rdf.pattern} filter An pattern for a set of bindings that might be added
to those in this {@link jQuery.rdf} object.
         * @param {Object} [options]
         * @param {Object} [options.namespaces] An object representing a set of namespace bindings used
to interpret any CURIEs within the pattern. Defaults to the namespace bindings defined on the {@link
jQuery.rdf#databank}.
         * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs
used within the pattern. Defaults to the base URI defined on the {@link jQuery.rdf#databank}.
         * @returns {jQuery.rdf} A new {@link jQuery.rdf} object whose {@link jQuery.rdf#parent} is this
{@link jQuery.rdf}.
         * @see jQuery.rdf#where
         * @see jQuery.rdf#filter
         * @see jQuery.rdf#about
         * @example
         * var rdf = rdf()
         *   .prefix('foaf', 'http://xmlns.com/foaf/0.1/')
         *   .prefix('rdf', 'http://www.w3.org/1999/02/22-rdf-syntax-ns#')
         *   .add('_:a  rdf:type        foaf:Person .')
         *   .add('_:a  foaf:name       "Alice" .')
         *   .add('_:a  foaf:mbox       &lt;mailto:alice@example.com> .')
         *   .add('_:a  foaf:mbox       &lt;mailto:alice@work.example> .')
         *   .add('_:b  rdf:type        foaf:Person .')
         *   .add('_:b  foaf:name       "Bob" .')
         *   .where('?x foaf:name ?name')
         *   .optional('?x foaf:mbox ?mbox');
         */
        optional: function (filter, options) {
            return this.where(filter, Utils.extend({}, options || {}, { optional: true }));
        },

        /**
         * Creates a new {@link jQuery.rdf} object whose set of bindings include <code>property</code>
```

```
and <code>value</code> for every triple that is about the specified resource.
         * @param {String|jQuery.rdf.resource} resource The subject of the matching triples.
         * @param {Object} [options]
         * @param {Object} [options.namespaces] An object representing a set of namespace bindings used
to interpret the resource if it's a CURIE. Defaults to the namespace bindings defined on the {@link
jQuery.rdf#databank}.
         * @param {String|jQuery.uri} [options.base] The base URI used to interpret the resource if it's
a relative URI (wrapped in <code>&lt;</code> and <code>&gt;</code>). Defaults to the base URI defined on
the {@link jQuery.rdf#databank}.
         * @returns {jQuery.rdf} A new {@link jQuery.rdf} object whose {@link jQuery.rdf#parent} is this
{@link jQuery.rdf}.
         * @see jQuery.rdf#where
         * @see jQuery.rdf#optional
         * @see jQuery.rdf#filter
         * @example
         * var rdf = rdf()
         *   .prefix('dc', ns.dc)
         *   .prefix('foaf', ns.foaf)
         *   .add('&lt;photo1.jpg> dc:creator &lt;http://www.blogger.com/profile/1109404> .')
         *   .add('&lt;http://www.blogger.com/profile/1109404> foaf:img &lt;photo1.jpg> .')
         *   .add('&lt;photo2.jpg> dc:creator &lt;http://www.blogger.com/profile/1109404> .')
         *   .add('&lt;http://www.blogger.com/profile/1109404> foaf:img &lt;photo2.jpg> .')
         *   .about('&lt;http://www.blogger.com/profile/1109404>');
         */
        about: function (resource, options) {
            return this.where(resource + ' ?property ?value', options);
        },

        /**
         * Creates a new {@link jQuery.rdf} object whose set of bindings include only those that satisfy
some arbitrary condition. There are two main ways to call this method: with two arguments in which case
the first is a binding to be tested and the second represents a condition on the test, or with one
argument which is a function that should return true for acceptable bindings.
         * @param {Function|String} property <p>In the two-argument version, this is the name of a
property to be tested against the condition specified in the second argument. In the one-argument
version, this is a function in which <code>this</code> is an object whose properties are a set of {@link
jQuery.rdf.resource}, {@link jQuery.rdf.literal} or {@link jQuery.rdf.blank} objects and whose arguments
are:</p>
         * <dl>
         *   <dt>i</dt>
         *   <dd>The index of the set of bindings amongst the other matches</dd>
         *   <dt>bindings</dt>
         *   <dd>An object representing the bindings (the same as <code>this</code>)</dd>
         *   <dt>triples</dt>
         *   <dd>The {@link jQuery.rdf.triple}s that underly this set of bindings</dd>
         * </dl>
         * @param {RegExp|String} condition In the two-argument version of this function, the condition
that the property's must match. If it is a regular expression, the value must match the regular
expression. If it is a {@link jQuery.rdf.literal}, the value of the literal must match the property's
value. Otherwise, they must be the same resource.
         * @returns {jQuery.rdf} A new {@link jQuery.rdf} object whose {@link jQuery.rdf#parent} is this
{@link jQuery.rdf}.
         * @see jQuery.rdf#where
         * @see jQuery.rdf#optional
         * @see jQuery.rdf#about
         * @example
         * var rdf = rdf()
         *   .prefix('foaf', 'http://xmlns.com/foaf/0.1/')
         *   .add('_:a foaf:surname "Jones" .')
         *   .add('_:b foaf:surname "Macnamara" .')
         *   .add('_:c foaf:surname "O\'Malley"')
         *   .add('_:d foaf:surname "MacFee"')
         *   .where('?person foaf:surname ?surname')
         *     .filter('surname', /^Ma?c/)
         *       .each(function () { scottish.push(this.surname.value); })
         *     .end()
         *     .filter('surname', /^O'/)
```

```
         *       .each(function () { irish.push(this.surname.value); })
         *     .end();
         * @example
         * var rdf = rdf()
         *   .prefix('foaf', 'http://xmlns.com/foaf/0.1/')
         *   .add('_:a foaf:surname "Jones" .')
         *   .add('_:b foaf:surname "Macnamara" .')
         *   .add('_:c foaf:surname "O\'Malley"')
         *   .add('_:d foaf:surname "MacFee"')
         *   .where('?person foaf:surname ?surname')
         *   .filter(function () { return this.surname !== "Jones"; })
         */
        filter: function (property, condition) {
            var func, query;
            if (typeof property === 'string') {
                if (condition.constructor === RegExp) {
                    /** @ignore func */
                    func = function () {
                        return condition.test(this[property].value);
                    };
                } else {
                    func = function () {
                        return this[property].type === 'literal' ? this[property].value === condition :
this[property] === condition;
                    };
                }
            } else {
                func = property;
            }
            query = rdf({ parent: this, filter: func });
            this.children.push(query);
            return query;
        },

        /**
         * Creates a new {@link jQuery.rdf} object containing one binding for each selected resource.
         * @param {String|Object} node The node to be selected. If this is a string beginning with a
question mark the resources are those identified by the bindings of that value in the currently selected
bindings. Otherwise, only the named resource is selected as the node.
         * @returns {jQuery.rdf} A new {@link jQuery.rdf} object.
         * @see jQuery.rdf#find
         * @see jQuery.rdf#back
         * @example
         * // returns an rdfQuery object with a pointer to <http://example.com/aReallyGreatBook>
         * var rdf = $('html').rdf()
         *   .node('<http://example.com/aReallyGreatBook>');
         */
        node: function (resource) {
            var variable, query;
            if (resource.toString().substring(0, 1) === '?') {
                variable = resource.toString().substring(1);
                query = rdf({ parent: this, navigate: variable });
            } else {
                if (typeof resource === 'string') {
                    resource = object(resource, { namespaces: this.prefix(), base: this.base() });
                }
                query = rdf({ parent: this, nodes: [resource] });
            }
            this.children.push(query);
            return query;
        },

        /**
         * Navigates from the resource identified by the 'node' binding to another node through the
property passed as the argument.
         * @param {String|Object} property The property whose value will be the new node.
         * @returns {jQuery.rdf} A new {@link jQuery.rdf} object whose {@link jQuery.rdf#parent} is this
```

```
{@link jQuery.rdf}.
         * @see jQuery.rdf#back
         * @see jQuery.rdf#node
         * @example
         * var creators = $('html').rdf()
         *   .node('<>')
         *   .find('dc:creator');
         */
        find: function (property) {
            return this.where('?node ' + property + ' ?object', { navigate: 'object' });
        },

        /**
         * Navigates from the resource identified by the 'node' binding to another node through the
property passed as the argument, like {jQuery.rdf#find}, but backwards.
         * @param {String|Object} property The property whose value will be the new node.
         * @returns {jQuery.rdf} A new {@link jQuery.rdf} object whose {@link jQuery.rdf#parent} is this
{@link jQuery.rdf}.
         * @see jQuery.rdf#find
         * @see jQuery.rdf#node
         * @example
         * var people = $('html').rdf()
         *   .node('foaf:Person')
         *   .back('rdf:type');
         */
        back: function (property) {
            return this.where('?subject ' + property + ' ?node', { navigate: 'subject' });
        },

        /**
         * Groups the bindings held by this {@link jQuery.rdf} object based on the values of the
variables passed as the parameter.
         * @param {String[]} [bindings] The variables to group by. The returned objects will contain all
their current properties, but those aside from the specified variables will be arrays listing the
relevant values.
         * @returns {jQuery} A jQuery object containing objects representing the grouped bindings.
         * @example
         * // returns one object per person and groups all the names and all the emails together in arrays
         * var grouped = rdf
         *   .where('?person foaf:name ?name')
         *   .where('?person foaf:email ?email')
         *   .group('person');
         * @example
         * // returns one object per surname/firstname pair, with the person property being an array in
the resulting objects
         * var grouped = rdf
         *   .where('?person foaf:first_name ?forename')
         *   .where('?person foaf:givenname ?surname')
         *   .group(['surname', 'forename']);
         */
        group: function (bindings) {
            var grouped = {}, results = [], i, key, v;
            if (!Utils.isArray(bindings)) {
                bindings = [bindings];
            }
            return group(this, bindings);
        },

        /**
         * Filters the variable bindings held by this {@link jQuery.rdf} object down to those listed in
the bindings parameter. This mirrors the <a href="http://www.w3.org/TR/rdf-sparql-query/#select">SELECT</
a> form in SPARQL.
         * @param {String[]} [bindings] The variables that you're interested in. The returned objects
will only contain those variables. If bindings is undefined, you will get all the variable bindings in
the returned objects.
         * @returns {Object[]} An array of objects with the properties named by the bindings parameter.
         * @example
```

```
         * var filtered = rdf
         *   .where('?photo dc:creator ?creator')
         *   .where('?creator foaf:img ?photo');
         * var selected = rdf.select(['creator']);
         */
        select: function (bindings) {
            var s = [], i, j;
            for (i = 0; i < this.length; i += 1) {
                if (bindings === undefined) {
                    s[i] = this[i];
                } else {
                    s[i] = {};
                    for (j = 0; j < bindings.length; j += 1) {
                        s[i][bindings[j]] = this[i][bindings[j]];
                    }
                }
            }
            return s;
        },

        /**
         * Provides <a href="http://n2.talis.com/wiki/
Bounded_Descriptions_in_RDF#Simple_Concise_Bounded_Description">simple concise bounded descriptions</a>
of the resources or bindings that are passed in the argument. This mirrors the <a href="http://www.w3.org/
TR/rdf-sparql-query/#describe">DESCRIBE</a> form in SPARQL.
         * @param {(String|jQuery.rdf.resource)[]} bindings An array that can contain strings, {@link
jQuery.rdf.resource}s or a mixture of the two. Any strings that begin with a question mark (<code>?</
code>) are taken as variable names; each matching resource is described by the function.
         * @returns {jQuery} A {@link jQuery} object that contains {@link jQuery.rdf.triple}s that
describe the listed resources.
         * @see jQuery.rdf.databank#describe
         * @example
         * rdf.dump($('html').rdf().describe(['<photo1.jpg>']));
         * @example
         * $('html').rdf()
         *   .where('?person foaf:img ?picture')
         *   .describe(['?photo'])
         */
        describe: function (bindings) {
            var i, j, binding, resources = [];
            for (i = 0; i < bindings.length; i += 1) {
                binding = bindings[i];
                if (binding.substring(0, 1) === '?') {
                    binding = binding.substring(1);
                    for (j = 0; j < this.length; j += 1) {
                        resources.push(this[j][binding]);
                    }
                } else {
                    resources.push(binding);
                }
            }
            return this.databank.describe(resources);
        },

        /**
         * Returns a new {@link jQuery.rdf} object that contains only one set of variable bindings. This
is designed to mirror the <a href="http://docs.jquery.com/Traversing/eq#index">jQuery#eq</a> method.
         * @param {Integer} n The index number of the match that should be selected.
         * @returns {jQuery.rdf} A new {@link jQuery.rdf} object with just that match.
         * @example
         * var rdf = rdf()
         *   .prefix('foaf', 'http://xmlns.com/foaf/0.1/')
         *   .add('_:a  foaf:name       "Alice" .')
         *   .add('_:a  foaf:homepage   <http://work.example.org/alice/> .')
         *   .add('_:b  foaf:name       "Bob" .')
         *   .add('_:b  foaf:mbox       <mailto:bob@work.example> .')
         *   .where('?x foaf:name ?name')
```

```
         *   .eq(1);
         */
        eq: function (n) {
            return this.filter(function (i) {
                return i === n;
            });
        },

        /**
         * Returns a {@link jQuery.rdf} object that includes no filtering (and therefore has no matches)
over the {@link jQuery.rdf#databank}.
         * @returns {jQuery.rdf} An empty {@link jQuery.rdf} object.
         * @example
         * $('html').rdf()
         *   .where('?person foaf:family_name "Hacker"')
         *   .where('?person foaf:givenname "Alice"')
         *   .each(...do something with Alice Hacker...)
         *   .reset()
         *   .where('?person foaf:family_name "Jones"')
         *   .where('?person foaf:givenname "Bob"')
         *   .each(...do something with Bob Jones...);
         */
        reset: function () {
            var query = this;
            while (query.parent !== undefined) {
                query = query.parent;
            }
            return query;
        },

        /**
         * Returns the parent {@link jQuery.rdf} object, which is equivalent to undoing the most recent
filtering operation (such as {@link jQuery.rdf#where} or {@link jQuery.rdf#filter}). This is designed to
mirror the <a href="http://docs.jquery.com/Traversing/end">jQuery#end</a> method.
         * @returns {jQuery.rdf}
         * @example
         * $('html').rdf()
         *   .where('?person foaf:family_name "Hacker"')
         *   .where('?person foaf:givenname "Alice"')
         *   .each(...do something with Alice Hacker...)
         *   .end()
         *   .where('?person foaf:givenname "Bob"')
         *   .each(...do something with Bob Hacker...);
         */
        end: function () {
            return this.parent;
        },

        /**
         * Returns the number of matches in this {@link jQuery.rdf} object (equivalent to {@link
jQuery.rdf#length}).
         * @returns {Integer} The number of matches in this {@link jQuery.rdf} object.
         * @see jQuery.rdf#length
         */
        size: function () {
            return this.length;
        },

        /**
         * Gets the triples that form the basis of the variable bindings that are the primary product of
{@link jQuery.rdf}. Getting hold of the triples can be useful for understanding the facts that form the
basis of the variable bindings.
         * @returns {jQuery} A {@link jQuery} object containing arrays of {@link jQuery.rdf.triple}
objects. A {@link jQuery} object is returned so that you can easily iterate over the contents.
         * @example
         * $('html').rdf()
         *   .where('?thing a foaf:Person')
```

```
         *   .sources()
         *   .each(function () {
         *     ...do something with the array of triples...
         *   });
         */
        sources: function () {
            return Utils.map(this.matches, function (match) {
                // return an array-of-an-array because arrays automatically get expanded by Utils.map()
                return match.triples;
            });
        },

        /**
         * Dumps the triples that form the basis of the variable bindings that are the primary product of
{@link jQuery.rdf} into a format that can be shown to the user or sent to a server.
         * @param {Object} [options] Options that control the formatting of the triples. See {@link
jQuery.rdf.dump} for details.
         * @see jQuery.rdf.dump
         */
        dump: function (options) {
            var triples = Utils.map(this.matches, function (match) {
                return match.triples;
            });
            options = Utils.extend({ namespaces: this.databank.namespaces, base: this.databank.base },
options || {});
            return rdf.dump(triples, options);
        },

        /**
         * Either returns the item specified by the argument or turns the {@link jQuery.rdf} object into
an array. This mirrors the <a href="http://docs.jquery.com/Core/get">jQuery#get</a> method.
         * @param {Integer} [num] The number of the item to be returned.
         * @returns {Object[]|Object} Returns either a single Object representing variable bindings or an
array of such.
         * @example
         * $('html').rdf()
         *   .where('?person a foaf:Person')
         *   .get(0)
         *   .subject
         *   .value;
         */
        get: function (num) {
            return (num === undefined) ? Utils.toArray(this) : this[num];
        },

        /**
         * Iterates over the matches held by the {@link jQuery.rdf} object and performs a function on
each of them. This mirrors the <a href="http://docs.jquery.com/Core/each">jQuery#each</a> method.
         * @param {Function} callback A function that is called for each match on the {@link jQuery.rdf}
object. Within the function, <code>this</code> is set to the object representing the variable bindings.
The function can take up to three parameters:
         * <dl>
         *   <dt>i</dt><dd>The index of the match amongst the other matches.</dd>
         *   <dt>bindings</dt><dd>An object representing the variable bindings for the match, the same as
<code>this</code>.</dd>
         *   <dt>triples</dt><dd>An array of {@link jQuery.rdf.triple}s associated with the particular
match.</dd>
         * </dl>
         * @returns {jQuery.rdf} The {@link jQuery.rdf} object.
         * @see jQuery.rdf#map
         * @example
         * var rdf = $('html').rdf()
         *   .where('?photo dc:creator ?creator')
         *   .where('?creator foaf:img ?photo')
         *   .each(function () {
         *     photos.push(this.photo.value);
         *   });
```

```
         */
        each: function (callback) {
            Utils.each(this.matches, function (match, i) {
                callback.call(match.bindings, match.bindings, i, match.triples);
            });
            return this;
        },

        /**
         * Iterates over the matches held by the {@link jQuery.rdf} object and creates a new {@link
jQuery} object that holds the result of applying the passed function to each one. This mirrors the <a
href="http://docs.jquery.com/Traversing/map">jQuery#map</a> method.
         * @param {Function} callback A function that is called for each match on the {@link jQuery.rdf}
object. Within the function, <code>this</code> is set to the object representing the variable bindings.
The function can take up to three parameters and should return some kind of value:
         * <dl>
         *   <dt>i</dt><dd>The index of the match amongst the other matches.</dd>
         *   <dt>bindings</dt><dd>An object representing the variable bindings for the match, the same as
<code>this</code>.</dd>
         *   <dt>triples</dt><dd>An array of {@link jQuery.rdf.triple}s associated with the particular
match.</dd>
         * </dl>
         * @returns {jQuery} A jQuery object holding the results of the function for each of the matches
on the original {@link jQuery.rdf} object.
         * @example
         * var photos = $('html').rdf()
         *   .where('?photo dc:creator ?creator')
         *   .where('?creator foaf:img ?photo')
         *   .map(function () {
         *     return this.photo.value;
         *   });
         */
        map: function (callback) {
            return Utils.flatten(Utils.map(this.matches, function (match, i) {
                // in the callback, "this" is the bindings, and the arguments are swapped from Utils.map()
                return callback.call(match.bindings, match.bindings, i, match.triples);
            }));
        }
    };

    rdf.fn.init.prototype = rdf.fn;

    rdf.gleaners = [];
    rdf.parsers = {};

    /**
     * Dumps the triples passed as the first argument into a format that can be shown to the user or sent
to a server.
     * @param {jQuery.rdf.triple[]} triples An array (or {@link jQuery} object) of {@link
jQuery.rdf.triple}s.
     * @param {Object} [options] Options that control the format of the dump.
     * @param {String} [options.format='application/json'] The mime type of the format of the dump. The
supported formats are:
     * <table>
     *   <tr><th>mime type</th><th>description</th></tr>
     *   <tr>
     *     <td><code>application/json</code></td>
     *     <td>An <a href="http://n2.talis.com/wiki/RDF_JSON_Specification">RDF/JSON</a> object</td>
     *   </tr>
     *   <tr>
     *     <td><code>application/rdf+xml</code></td>
     *     <td>An DOMDocument node holding XML in <a href="http://www.w3.org/TR/rdf-syntax-grammar/">RDF/
XML syntax</a></td>
     *   </tr>
     *   <tr>
     *     <td><code>text/turtle</code></td>
     *     <td>A String holding a representation of the RDF in <a href="http://www.w3.org/TeamSubmission/
```

```
turtle/">Turtle syntax</a></td>
     *   </tr>
     * </table>
     * @param {Object} [options.namespaces={}] A set of namespace bindings used when mapping resource
URIs to CURIEs or QNames (particularly in a RDF/XML serialisation).
     * @param {boolean} [options.serialize=false] If true, rather than creating an Object, the function
will return a string which is ready to display or send to a server.
     * @param {boolean} [options.indent=false] If true, the serialised (RDF/XML) output has indentation
added to it to make it more readable.
     * @returns {Object|String} The alternative representation of the triples.
     */
    rdf.dump = function (triples, options) {
        var opts = Utils.extend({}, rdf.dump.defaults, options || {}),
            format = opts.format,
            serialize = opts.serialize,
            dump, parser, parsers;
        parser = rdf.parsers[format];
        if (parser === undefined) {
            parsers = [];
            Utils.each(rdf.parsers, function (p) {
                parsers.push(p);
            });
            throw "Unrecognised dump format: " + format + ". Expected one of " + parsers.join(", ");
        }
        dump = parser.dump(triples, opts);
        return serialize ? parser.serialize(dump) : dump;
    };

    rdf.dump.defaults = {
        format: 'application/json',
        serialize: false,
        indent: false,
        namespaces: {}
    }

    /**
     * <p>Creates a new jQuery.rdf.databank object. This should be invoked as a method rather than
constructed using new; indeed you will not usually want to generate these objects directly, but
manipulate them through a {@link jQuery.rdf} object.</p>
     * @class Represents a triplestore, holding a bunch of {@link jQuery.rdf.triple}s.
     * @param {(String|jQuery.rdf.triple)[]} [triples=[]] An array of triples to store in the databank.
     * @param {Object} [options] Initialisation of the databank.
     * @param {Object} [options.namespaces] An object representing a set of namespace bindings used when
interpreting the CURIEs in strings representing triples. Rather than passing this in when you construct
the {@link jQuery.rdf.databank} instance, you will usually want to use the {@link
jQuery.rdf.databank#prefix} method.
     * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs used
within the strings representing triples.
     * @returns {jQuery.rdf.databank} The newly-created databank.
     * @see jQuery.rdf
     */
    rdf.databank = function (triples, options) {
        return new rdf.databank.fn.init(triples, options);
    };

    rdf.databank.fn = rdf.databank.prototype = {
        init: function (triples, options) {
            var i;
            triples = triples || [];
            options = options || {};
            this.id = databankID();
            databanks[this.id] = this;
            if (options.union === undefined) {
                this.queries = {};
                this.tripleStore = [];
                this.subjectIndex = {};
                this.propertyIndex = {};
```

```
                this.objectIndex = {};
                this.baseURI = options.base || URI.base();
                this.namespaces = Utils.extend({}, options.namespaces || {});
                for (i = 0; i < triples.length; i += 1) {
                    this.add(triples[i]);
                }
            } else {
                this.union = options.union;
            }
            return this;
        },

        /**
         * Sets or returns the base URI of the {@link jQuery.rdf.databank}.
         * @param {String|jQuery.uri} [base]
         * @returns A {@link jQuery.uri} if no base URI is specified, otherwise returns this {@link
jQuery.rdf.databank} object.
         * @see jQuery.rdf#base
         */
        base: function (base) {
            if (this.union === undefined) {
                if (base === undefined) {
                    return this.baseURI;
                } else {
                    this.baseURI = base;
                    return this;
                }
            } else if (base === undefined) {
                return this.union[0].base();
            } else {
                Utils.each(this.union, function (databank, i) {
                    databank.base(base);
                });
                return this;
            }
        },

        /**
         * Sets or returns a namespace binding on the {@link jQuery.rdf.databank}.
         * @param {String} [prefix]
         * @param {String} [namespace]
         * @returns {Object|jQuery.uri|jQuery.rdf} If no prefix or namespace is specified, returns an
object providing all namespace bindings on the {@link jQuery.rdf#databank}. If a prefix is specified
without a namespace, returns the {@link jQuery.uri} associated with that prefix. Otherwise returns this
{@link jQuery.rdf} object after setting the namespace binding.
         * @see jQuery.rdf#prefix
         */
        prefix: function (prefix, uri) {
            var namespaces = {};
            if (this.union === undefined) {
                if (prefix === undefined) {
                    return this.namespaces;
                } else if (uri === undefined) {
                    return this.namespaces[prefix];
                } else {
                    this.namespaces[prefix] = uri;
                    return this;
                }
            } else if (uri === undefined) {
                Utils.each(this.union, function (databank, i) {
                    Utils.extend(namespaces, databank.prefix());
                });
                if (prefix === undefined) {
                    return namespaces;
                } else {
                    return namespaces[prefix];
                }
```

```
            } else {
                Utils.each(this.union, function (databank) {
                    databank.prefix(prefix, uri);
                });
                return this;
            }
        },

        /**
         * Adds a triple to the {@link jQuery.rdf.databank} or another {@link jQuery.rdf.databank} object
to create a union.
         * @param {String|jQuery.rdf.triple|jQuery.rdf.databank} triple The triple or {@link
jQuery.rdf.databank} object to be added to this one. If the triple is a {@link jQuery.rdf.databank}
object, the two databanks are unioned together. If the triple is a string, it's parsed as a {@link
jQuery.rdf.triple}.
         * @param {Object} [options]
         * @param {Object} [options.namespaces] An object representing a set of namespace bindings used
to interpret CURIEs within the triple. Defaults to the namespace bindings defined on the {@link
jQuery.rdf.databank}.
         * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs
used within the triple. Defaults to the base URI defined on the {@link jQuery.rdf.databank}.
         * @param {Integer} [options.depth] The number of links to traverse to gather more information
about the subject, property and object of the triple.
         * @returns {jQuery.rdf.databank} This {@link jQuery.rdf.databank} object.
         * @see jQuery.rdf#add
         */
        add: function (triple, options) {
            var base = (options && options.base) || this.base(),
                namespaces = Utils.extend({}, this.prefix(), (options && options.namespaces) || {}),
                depth = (options && options.depth) || rdf.databank.defaults.depth,
                proxy = (options && options.proxy) || rdf.databank.defaults.proxy,
                databank;
            if (triple === this) {
                return this;
            } else if (triple.subjectIndex !== undefined) {
                // merging two databanks
                if (this.union === undefined) {
                    databank = rdf.databank(undefined, { union: [this, triple] });
                    return databank;
                } else {
                    this.union.push(triple);
                    return this;
                }
            } else {
                if (typeof triple === 'string') {
                    triple = rdf.triple(triple, { namespaces: namespaces, base: base, source: triple });
                }
                if (this.union === undefined) {
                    if (this.subjectIndex[triple.subject] === undefined) {
                        this.subjectIndex[triple.subject] = [];
                        if (depth > 0 && triple.subject.type === 'uri') {
                            this.load(triple.subject.value, { depth: depth - 1, proxy: proxy });
                        }
                    }
                    if (this.propertyIndex[triple.property] === undefined) {
                        this.propertyIndex[triple.property] = [];
                        if (depth > 0) {
                            this.load(triple.property.value, { depth: depth - 1, proxy: proxy });
                        }
                    }
                    if (Utils.indexOf(this.subjectIndex[triple.subject], triple) === -1) {
                        this.tripleStore.push(triple);
                        this.subjectIndex[triple.subject].push(triple);
                        this.propertyIndex[triple.property].push(triple);
                        if (triple.object.type === 'uri' || triple.object.type === 'bnode') {
                            if (this.objectIndex[triple.object] === undefined) {
                                this.objectIndex[triple.object] = [];
```

```
                            if (depth > 0 && triple.object.type === 'uri') {
                                this.load(triple.object.value, { depth: depth - 1, proxy: proxy });
                            }
                        }
                        this.objectIndex[triple.object].push(triple);
                    }
                    addToDatabankQueries(this, triple);
                }
            } else {
                Utils.each(this.union, function (databank) {
                    databank.add(triple);
                });
            }
            return this;
        }
    },

    /**
     * Removes a triple from the {@link jQuery.rdf.databank}.
     * @param {String|jQuery.rdf.triple} triple The triple to be removed.
     * @param {Object} [options]
     * @param {Object} [options.namespaces] An object representing a set of namespace bindings used to interpret any CURIEs within the triple. Defaults to the namespace bindings defined on the {@link jQuery.rdf.databank}.
     * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs used within the triple. Defaults to the base URI defined on the {@link jQuery.rdf.databank}.
     * @returns {jQuery.rdf.databank} The {@link jQuery.rdf.databank} object itself.
     * @see jQuery.rdf#remove
     */
    remove: function (triple, options) {
        var base = (options && options.base) || this.base(),
            namespaces = Utils.extend({}, this.prefix(), (options && options.namespaces) || {}),
            striples, ptriples, otriples,
            databank;
        if (typeof triple === 'string') {
            triple = rdf.triple(triple, { namespaces: namespaces, base: base, source: triple });
        }
        this.tripleStore.splice(Utils.indexOf(this.tripleStore, triple), 1);
        striples = this.subjectIndex[triple.subject];
        if (striples !== undefined) {
            striples.splice(Utils.indexOf(striples, triple), 1);
        }
        ptriples = this.propertyIndex[triple.property];
        if (ptriples !== undefined) {
            ptriples.splice(Utils.indexOf(ptriples, triple), 1);
        }
        if (triple.object.type === 'uri' || triple.object.type === 'bnode') {
            otriples = this.objectIndex[triple.object];
            if (otriples !== undefined) {
                otriples.splice(Utils.indexOf(otriples, triple), 1);
            }
        }
        removeFromDatabankQueries(this, triple);
        return this;
    },

    /**
     * Creates a new databank containing all the triples in this {@link jQuery.rdf.databank} except those in the {@link jQuery.rdf.databank} passed as the argument.
     * @param {jQuery.rdf.databank} data The other {@link jQuery.rdf.databank}
     * @returns {jQuery.rdf.databank} A new {@link jQuery.rdf.databank} containing the triples in this {@link jQuery.rdf.databank} except for those in the data parameter.
     * @example
     * var old = $('html').rdf().databank;
     * ...some processing occurs...
     * var new = $('html').rdf().databank;
     * var added = new.except(old);
```

```
         * var removed = old.except(new);
         */
        except: function (data) {
            var store = data.subjectIndex,
                diff = [];
            Utils.each(this.subjectIndex, function (ts, s) {
                var ots = store[s];
                if (ots === undefined) {
                    diff = diff.concat(ts);
                } else {
                    Utils.each(ts, function (t) {
                        if (Utils.indexOf(ots, t) === -1) {
                            diff.push(t);
                        }
                    });
                }
            });
            return rdf.databank(diff);
        },

        /**
         * Provides a {@link jQuery} object containing the triples held in this {@link
jQuery.rdf.databank}.
         * @returns {jQuery} A {@link jQuery} object containing {@link jQuery.rdf.triple} objects.
         */
        triples: function () {
            var s, triples = [];
            if (this.union === undefined) {
                triples = this.tripleStore;
            } else {
                Utils.each(this.union, function (databank, i) {
                    triples = triples.concat(databank.triples());
                });
                triples = unique(triples);
            }
            return triples;
        },

        /**
         * Tells you how many triples the databank contains.
         * @returns {Integer} The number of triples in the {@link jQuery.rdf.databank}.
         * @example $('html').rdf().databank.size();
         */
        size: function () {
            return this.triples().length;
        },

        /**
         * Provides <a href="http://n2.talis.com/wiki/
Bounded_Descriptions_in_RDF#Simple_Concise_Bounded_Description">simple concise bounded descriptions</a>
of the resources that are passed in the argument. This mirrors the <a href="http://www.w3.org/TR/rdf-
sparql-query/#describe">DESCRIBE</a> form in SPARQL.
         * @param {(String|jQuery.rdf.resource)[]} resources An array that can contain strings, {@link
jQuery.rdf.resource}s or a mixture of the two.
         * @returns {jQuery} A {@link jQuery} object holding the {@link jQuery.rdf.triple}s that describe
the listed resources.
         * @see jQuery.rdf#describe
         */
        describe: function (resources) {
            var i, r, t, rhash = {}, triples = [];
            while (resources.length > 0) {
                r = resources.pop();
                if (rhash[r] === undefined) {
                    if (r.value === undefined) {
                        r = rdf.resource(r);
                    }
                    if (this.subjectIndex[r] !== undefined) {
```

```
                        for (i = 0; i < this.subjectIndex[r].length; i += 1) {
                            t = this.subjectIndex[r][i];
                            triples.push(t);
                            if (t.object.type === 'bnode') {
                                resources.push(t.object);
                            }
                        }
                    }
                    if (this.objectIndex[r] !== undefined) {
                        for (i = 0; i < this.objectIndex[r].length; i += 1) {
                            t = this.objectIndex[r][i];
                            triples.push(t);
                            if (t.subject.type === 'bnode') {
                                resources.push(t.subject);
                            }
                        }
                    }
                    rhash[r] = true;
                }
            }
            return unique(triples);
        },

        /**
         * Dumps the triples in the databank into a format that can be shown to the user or sent to a
server.
         * @param {Object} [options] Options that control the formatting of the triples. See {@link
jQuery.rdf.dump} for details.
         * @returns {Object|Node|String}
         * @see jQuery.rdf.dump
         */
        dump: function (options) {
            options = Utils.extend({ namespaces: this.namespaces, base: this.base }, options || {});
            return rdf.dump(this.triples(), options);
        },

        /**
         * Loads some data into the databank.
         * @param {Node|Object|String} data If the data is a string and starts with 'http://' then it's
taken to be a URI and data is loaded from that URI via the proxy specified in the options. If it doesn't
start with 'http://' then it's taken to be a serialized version of some format capable of representing
RDF, parsed and interpreted. If the data is a node, it's interpreted to be an <a href="http://www.w3.org/
TR/rdf-syntax-grammar/">RDF/XML syntax</a> document and will be parsed as such. Otherwise, it's taken to
be a <a href="http://n2.talis.com/wiki/RDF_JSON_Specification">RDF/JSON</a> object.
         * @param {Object} opts Options governing the loading of the data.
         * @param {String} [opts.format] The mime type of the format the data is in, particularly useful
if you're supplying the data as a string. If unspecified, the data will be sniffed to see if it might be
HTML, RDF/XML, RDF/JSON or Turtle.
         * @param {boolean} [opts.async=true] When loading data from a URI, this determines whether it
will be done synchronously or asynchronously.
         * @param {Function} [opts.success] When loading data from a URI, a function that will be called
after the data is successfully loaded.
         * @param {Function} [opts.error] When loading data from a URI, a function that will be called if
there's an error when accessing the URI.
         * @param {String} [opts.proxy='http://www.jenitennison.com/rdfquery/proxy.php'] The URI for a
server-side proxy through which the data can be accessed. This does not have to be hosted on the same
server as this Javascript, the HTML page or the remote data. The proxy must accept id, url and depth
parameters and respond with some Javascript that will invoke the {@link jQuery.rdf.databank.load}
function. <a href="http://code.google.com/p/rdfquery/source/browse/#svn/trunk/proxies">Example proxies</
a> that do the right thing are available. If you are intending to use this facility a lot, please do not
use the default proxy.
         * @param {integer} [opts.depth=0] Triggers recursive loading of located resources, to the depth
specified. This is useful for automatically populating a databank with linked data.
         * @returns {jQuery.rdf.databank} The {@link jQuery.rdf.databank} itself.
         * @see jQuery.rdf#load
         */
        load: function (data, opts) {
```

```
            var i, triples, url, script, parser, docElem,
                format = (opts && opts.format),
                async = (opts && opts.async) || rdf.databank.defaults.async,
                success = (opts && opts.success) || rdf.databank.defaults.success,
                error = (opts && opts.error) || rdf.databank.defaults.error,
                proxy = (opts && opts.proxy) || rdf.databank.defaults.proxy,
                depth = (opts && opts.depth) || rdf.databank.defaults.depth;
            url = (typeof data === 'string' && data.substring(1, 7) === 'http://') ? URI(data) : data;
            if (url.scheme) {
                //console.log("DO SOMETHING HERE");
                /*
                 if (!queue(this, url, { success: success, error: error })) {
                 script = '<script type="text/javascript" src="' + proxy + '?id=' + this.id +
'&amp;depth=' + depth + '&amp;url=' + encodeURIComponent(url.resolve('').toString()) + '"></script>';
                 if (async) {
                 setTimeout("$('head').append('" + script + "')", 0);
                 } else {
                 $('head').append(script);
                 }
                 }
                 */
                return this;
            } else {
                if (format === undefined) {
                    if (typeof data === 'string') {
                        if (data.substring(0, 1) === '{') {
                            format = 'application/json';
                        } else if (data.substring(0, 14) === '<!DOCTYPE html' || data.substring(0, 5) ===
'<html') {
                            format = 'application/xhtml+xml';
                        } else if (data.substring(0, 5) === '<?xml' || data.substring(0, 8) ===
'<rdf:RDF') {
                            format = 'application/rdf+xml';
                        } else {
                            format = 'text/turtle';
                        }
                    } else if (data.documentElement || data.ownerDocument) {
                        docElem = data.documentElement ? data.documentElement :
data.ownerDocument.documentElement;
                        if (docElem.nodeName === 'html') {
                            format = 'application/xhtml+xml';
                        } else {
                            format = 'application/rdf+xml';
                        }
                    } else {
                        format = 'application/json';
                    }
                }
                parser = rdf.parsers[format];
                if (typeof data === 'string') {
                    data = parser.parse(data);
                }
                triples = parser.triples(data);
                for (i = 0; i < triples.length; i += 1) {
                    this.add(triples[i], opts);
                }
                return this;
            }
        },

        /**
         * Provides a string representation of the databank which simply specifies how many triples it
contains.
         * @returns {String}
         */
        toString: function () {
            return '[Databank with ' + this.size() + ' triples]';
```

```
        }
    };

    rdf.databank.fn.init.prototype = rdf.databank.fn;

    rdf.databank.defaults = {
        parse: false,
        async: true,
        success: null,
        error: null,
        depth: 0,
        proxy: 'http://www.jenitennison.com/rdfquery/proxy.php'
    };

    rdf.databank.load = function (id, url, doc, opts) {
        if (doc !== undefined) {
            databanks[id].load(doc, opts);
        }
        dequeue(databanks[id], url, (doc === undefined) ? 'error' : 'success', opts);
    };

    /**
     * <p>Creates a new jQuery.rdf.pattern object. This should be invoked as a method rather than
constructed using new; indeed you will not usually want to generate these objects directly, since they
are automatically created from strings where necessary, such as by {@link jQuery.rdf#where}.</p>
     * @class Represents a pattern that may or may not match a given {@link jQuery.rdf.triple}.
     * @param {String|jQuery.rdf.resource|jQuery.rdf.blank} subject The subject pattern, or a single
string that defines the entire pattern. If the subject is specified as a string, it can be a fixed
resource (<code>&lt;<var>uri</var>&gt;</code> or <code><var>curie</var></code>), a blank node
(<code>_:<var>id</var></code>) or a variable placeholder (<code>?<var>name</var></code>).
     * @param {String|jQuery.rdf.resource} [property] The property pattern. If the property is specified
as a string, it can be a fixed resource (<code>&lt;<var>uri</var>&gt;</code> or <code><var>curie</var></
code>) or a variable placeholder (<code>?<var>name</var></code>).
     * @param {String|jQuery.rdf.resource|jQuery.rdf.blank|jQuery.rdf.literal} [value] The value pattern.
If the property is specified as a string, it can be a fixed resource (<code>&lt;<var>uri</var>&gt;</code>
or <code><var>curie</var></code>), a blank node (<code>_:<var>id</var></code>), a literal
(<code>"<var>value</var>"</code>) or a variable placeholder (<code>?<var>name</var></code>).
     * @param {Object} [options] Initialisation of the pattern.
     * @param {Object} [options.namespaces] An object representing a set of namespace bindings used when
interpreting the CURIEs in the subject, property and object.
     * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs used
within the subject, property and object.
     * @param {boolean} [options.optional]
     * @returns {jQuery.rdf.pattern} The newly-created pattern.
     * @throws {String} Errors if any of the strings are not in a recognised format.
     * @example pattern = rdf.pattern('?person', rdf.type, 'foaf:Person', { namespaces: { foaf: "http://
xmlns.com/foaf/0.1/" }});
     * @example
     * pattern = rdf.pattern('?person a foaf:Person', {
     *   namespaces: { foaf: "http://xmlns.com/foaf/0.1/" },
     *   optional: true
     * });
     * @see jQuery.rdf#where
     * @see jQuery.rdf.resource
     * @see jQuery.rdf.blank
     * @see jQuery.rdf.literal
     */
    rdf.pattern = function (subject, property, object, options) {
        var pattern, m, optional;
        // using a two-argument version; first argument is a Turtle statement string
        if (object === undefined) {
            options = property || {};
            m = Utils.trim(subject).match(tripleRegex);
            if (m.length === 3 || (m.length === 4 && m[3] === '.')) {
                subject = m[0];
                property = m[1];
                object = m[2];
```

```
            } else {
                throw "Bad Pattern: Couldn't parse string " + subject;
            }
            optional = (options.optional === undefined) ? rdf.pattern.defaults.optional :
options.optional;
        }
        if (memPattern[subject] &&
            memPattern[subject][property] &&
            memPattern[subject][property][object] &&
            memPattern[subject][property][object][optional]) {
            return memPattern[subject][property][object][optional];
        }
        pattern = new rdf.pattern.fn.init(subject, property, object, options);
        if (memPattern[pattern.subject] &&
            memPattern[pattern.subject][pattern.property] &&
            memPattern[pattern.subject][pattern.property][pattern.object] &&
            memPattern[pattern.subject][pattern.property][pattern.object][pattern.optional]) {
            return memPattern[pattern.subject][pattern.property][pattern.object][pattern.optional];
        } else {
            if (memPattern[pattern.subject] === undefined) {
                memPattern[pattern.subject] = {};
            }
            if (memPattern[pattern.subject][pattern.property] === undefined) {
                memPattern[pattern.subject][pattern.property] = {};
            }
            if (memPattern[pattern.subject][pattern.property][pattern.object] === undefined) {
                memPattern[pattern.subject][pattern.property][pattern.object] = {};
            }
            memPattern[pattern.subject][pattern.property][pattern.object][pattern.optional] = pattern;
            return pattern;
        }
    };

    rdf.pattern.fn = rdf.pattern.prototype = {
        init: function (s, p, o, options) {
            var opts = Utils.extend({}, rdf.pattern.defaults, options);
            /**
             * The placeholder for the subject of triples matching against this pattern.
             * @type String|jQuery.rdf.resource|jQuery.rdf.blank
             */
            this.subject = s.toString().substring(0, 1) === '?' ? s : subject(s, opts);
            /**
             * The placeholder for the property of triples matching against this pattern.
             * @type String|jQuery.rdf.resource
             */
            this.property = p.toString().substring(0, 1) === '?' ? p : property(p, opts);
            /**
             * The placeholder for the object of triples matching against this pattern.
             * @type String|jQuery.rdf.resource|jQuery.rdf.blank|jQuery.rdf.literal
             */
            this.object = o.toString().substring(0, 1) === '?' ? o : object(o, opts);
            /**
             * Whether the pattern should only optionally match against the triple
             * @type boolean
             */
            this.optional = opts.optional;
            return this;
        },

        /**
         * Creates a new {@link jQuery.rdf.pattern} with any variable placeholders within this one's
subject, property or object filled in with values from the bindings passed as the argument.
         * @param {Object} bindings An object holding the variable bindings to be used to replace any
placeholders in the pattern. These bindings are of the type held by the {@link jQuery.rdf} object.
         * @returns {jQuery.rdf.pattern} A new {@link jQuery.rdf.pattern} object.
         * @example
         * pattern = rdf.pattern('?thing a ?class');
```

```
         * // pattern2 matches all triples that indicate the classes of this page.
         * pattern2 = pattern.fill({ thing: rdf.resource('<>') });
         */
        fill: function (bindings) {
            var s = this.subject,
                p = this.property,
                o = this.object;
            if (typeof s === 'string' && bindings[s.substring(1)]) {
                s = bindings[s.substring(1)];
            }
            if (typeof p === 'string' && bindings[p.substring(1)]) {
                p = bindings[p.substring(1)];
            }
            if (typeof o === 'string' && bindings[o.substring(1)]) {
                o = bindings[o.substring(1)];
            }
            return rdf.pattern(s, p, o, { optional: this.optional });
        },

        /**
         * Creates a new Object holding variable bindings by matching the passed triple against this
pattern.
         * @param {jQuery.rdf.triple} triple A {@link jQuery.rdf.triple} for this pattern to match
against.
         * @returns {null|Object} An object containing the bindings of variables (as specified in this
pattern) to values (as specified in the triple), or <code>null</code> if the triple doesn't match the
pattern.
         * pattern = rdf.pattern('?thing a ?class');
         * bindings = pattern.exec(rdf.triple('<> a foaf:Person', { namespaces: ns }));
         * thing = bindings.thing; // the resource for this page
         * class = bindings.class; // a resource for foaf:Person
         */
        exec: function (triple) {
            var binding = {};
            binding = testResource(triple.subject, this.subject, binding);
            if (binding === null) {
                return null;
            }
            binding = testResource(triple.property, this.property, binding);
            if (binding === null) {
                return null;
            }
            binding = testResource(triple.object, this.object, binding);
            return binding;
        },

        /**
         * Tests whether this pattern has any variable placeholders in it or not.
         * @returns {boolean} True if the pattern doesn't contain any variable placeholders.
         * @example
         * rdf.pattern('?thing a ?class').isFixed(); // false
         * rdf.pattern('<> a foaf:Person', { namespaces: ns }).isFixed(); // true
         */
        isFixed: function () {
            return typeof this.subject !== 'string' &&
                typeof this.property !== 'string' &&
                typeof this.object !== 'string';
        },

        /**
         * Creates a new triple based on the bindings passed to the pattern, if possible.
         * @param {Object} bindings An object holding the variable bindings to be used to replace any
placeholders in the pattern. These bindings are of the type held by the {@link jQuery.rdf} object.
         * @returns {null|jQuery.rdf.triple} A new {@link jQuery.rdf.triple} object, or null if not all
the variable placeholders in the pattern are specified in the bindings. The {@link
jQuery.rdf.triple#source} of the generated triple is set to the string value of this pattern.
         * @example
```

```
         * pattern = rdf.pattern('?thing a ?class');
         * // triple is a new triple '<> a foaf:Person'
         * triple = pattern.triple({
         *   thing: rdf.resource('<>'),
         *   class: rdf.resource('foaf:Person', { namespaces: ns })
         * });
         */
        triple: function (bindings) {
            var t = this;
            if (!this.isFixed()) {
                t = this.fill(bindings);
            }
            if (t.isFixed()) {
                return rdf.triple(t.subject, t.property, t.object, { source: this.toString() });
            } else {
                return null;
            }
        },

        /**
         * Returns a string representation of the pattern by concatenating the subject, property and
object.
         * @returns {String}
         */
        toString: function () {
            return this.subject + ' ' + this.property + ' ' + this.object;
        }
    };

    rdf.pattern.fn.init.prototype = rdf.pattern.fn;

    rdf.pattern.defaults = {
        base: URI.base(),
        namespaces: {},
        optional: false
    };

    /**
     * <p>Creates a new jQuery.rdf.triple object. This should be invoked as a method rather than
constructed using new; indeed you will not usually want to generate these objects directly, since they
are automatically created from strings where necessary, such as by {@link jQuery.rdf#add}.</p>
     * @class Represents an RDF triple.
     * @param {String|jQuery.rdf.resource|jQuery.rdf.blank} subject The subject of the triple, or a
single string that defines the entire triple. If the subject is specified as a string, it can be a fixed
resource (<code>&lt;<var>uri</var>&gt;</code> or <code><var>curie</var></code>) or a blank node
(<code>_:<var>id</var></code>).
     * @param {String|jQuery.rdf.resource} [property] The property pattern. If the property is specified
as a string, it must be a fixed resource (<code>&lt;<var>uri</var>&gt;</code> or <code><var>curie</var></
code>).
     * @param {String|jQuery.rdf.resource|jQuery.rdf.blank|jQuery.rdf.literal} [value] The value pattern.
If the property is specified as a string, it can be a fixed resource (<code>&lt;<var>uri</var>&gt;</code>
or <code><var>curie</var></code>), a blank node (<code>_:<var>id</var></code>), or a literal
(<code>"<var>value</var>"</code>).
     * @param {Object} [options] Initialisation of the triple.
     * @param {Object} [options.namespaces] An object representing a set of namespace bindings used when
interpreting the CURIEs in the subject, property and object.
     * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs used
within the subject, property and object.
     * @returns {jQuery.rdf.triple} The newly-created triple.
     * @throws {String} Errors if any of the strings are not in a recognised format.
     * @example pattern = rdf.triple('<>', rdf.type, 'foaf:Person', { namespaces: { foaf: "http://
xmlns.com/foaf/0.1/" }});
     * @example
     * pattern = rdf.triple('<> a foaf:Person', {
     *   namespaces: { foaf: "http://xmlns.com/foaf/0.1/" }
     * });
     * @see jQuery.rdf#add
```

```
     * @see jQuery.rdf.resource
     * @see jQuery.rdf.blank
     * @see jQuery.rdf.literal
     */
    rdf.triple = function (subject, property, object, options) {
        var triple, graph, m;
        // using a two-argument version; first argument is a Turtle statement string
        if (object === undefined) {
            options = property;
            m = Utils.trim(subject).match(tripleRegex);
            if (m.length === 3 || (m.length === 4 && m[3] === '.')) {
                subject = m[0];
                property = m[1];
                object = m[2];
            } else {
                throw "Bad Triple: Couldn't parse string " + subject;
            }
        }
        graph = (options && options.graph) || '';
        if (memTriple[graph] &&
            memTriple[graph][subject] &&
            memTriple[graph][subject][property] &&
            memTriple[graph][subject][property][object]) {
            return memTriple[graph][subject][property][object];
        }
        triple = new rdf.triple.fn.init(subject, property, object, options);
        graph = triple.graph || '';
        if (memTriple[graph] &&
            memTriple[graph][triple.subject] &&
            memTriple[graph][triple.subject][triple.property] &&
            memTriple[graph][triple.subject][triple.property][triple.object]) {
            return memTriple[graph][triple.subject][triple.property][triple.object];
        } else {
            if (memTriple[graph] === undefined) {
                memTriple[graph] = {};
            }
            if (memTriple[graph][triple.subject] === undefined) {
                memTriple[graph][triple.subject] = {};
            }
            if (memTriple[graph][triple.subject][triple.property] === undefined) {
                memTriple[graph][triple.subject][triple.property] = {};
            }
            memTriple[graph][triple.subject][triple.property][triple.object] = triple;
            return triple;
        }
    };

    rdf.triple.fn = rdf.triple.prototype = {
        init: function (s, p, o, options) {
            var opts;
            opts = Utils.extend({}, rdf.triple.defaults, options);
            /**
             * The subject of the triple.
             * @type jQuery.rdf.resource|jQuery.rdf.blank
             */
            this.subject = subject(s, opts);
            /**
             * The property of the triple.
             * @type jQuery.rdf.resource
             */
            this.property = property(p, opts);
            /**
             * The object of the triple.
             * @type jQuery.rdf.resource|jQuery.rdf.blank|jQuery.rdf.literal
             */
            this.object = object(o, opts);
            /**
```

```
             * (Experimental) The named graph the triple belongs to.
             * @type jQuery.rdf.resource|jQuery.rdf.blank
             */
            this.graph = opts.graph === undefined ? undefined : subject(opts.graph, opts);
            /**
             * The source of the triple, which might be a node within the page (if the RDF is generated
from the page) or a string holding the pattern that generated the triple.
             */
            this.source = opts.source;
            return this;
        },

        /**
         * Always returns true for triples.
         * @see jQuery.rdf.pattern#isFixed
         */
        isFixed: function () {
            return true;
        },

        /**
         * Always returns this triple.
         * @see jQuery.rdf.pattern#triple
         */
        triple: function (bindings) {
            return this;
        },

        /**
         * Returns a <a href="http://n2.talis.com/wiki/RDF_JSON_Specification">RDF/JSON</a>
representation of this triple.
         * @returns {Object}
         */
        dump: function () {
            var e = {},
                s = this.subject.value.toString(),
                p = this.property.value.toString();
            e[s] = {};
            e[s][p] = this.object.dump();
            return e;
        },

        /**
         * Returns a string representing this triple in Turtle format.
         * @returns {String}
         */
        toString: function () {
            return this.subject + ' ' + this.property + ' ' + this.object + ' .';
        }
    };

    rdf.triple.fn.init.prototype = rdf.triple.fn;

    rdf.triple.defaults = {
        base: URI.base(),
        source: [],
        namespaces: {}
    };

    /**
     * <p>Creates a new jQuery.rdf.resource object. This should be invoked as a method rather than
constructed using new; indeed you will not usually want to generate these objects directly, since they
are automatically created from strings where necessary, such as by {@link jQuery.rdf#add}.</p>
     * @class Represents an RDF resource.
     * @param {String|jQuery.uri} value The value of the resource. If it's a string it must be in the
format <code>&lt;<var>uri</var>&gt;</code> or <code><var>curie</var></code>.
     * @param {Object} [options] Initialisation of the resource.
```

```
     * @param {Object} [options.namespaces] An object representing a set of namespace bindings used when
interpreting the CURIE specifying the resource.
     * @param {String|jQuery.uri} [options.base] The base URI used to interpret any relative URIs used
within the URI specifying the resource.
     * @returns {jQuery.rdf.resource} The newly-created resource.
     * @throws {String} Errors if the string is not in a recognised format.
     * @example thisPage = rdf.resource('<>');
     * @example foaf.Person = rdf.resource('foaf:Person', { namespaces: ns });
     * @see jQuery.rdf.pattern
     * @see jQuery.rdf.triple
     * @see jQuery.rdf.blank
     * @see jQuery.rdf.literal
     */
    rdf.resource = function (value, options) {
        var resource;
        if (memResource[value]) {
            return memResource[value];
        }
        resource = new rdf.resource.fn.init(value, options);
        if (memResource[resource]) {
            return memResource[resource];
        } else {
            memResource[resource] = resource;
            return resource;
        }
    };

    rdf.resource.fn = rdf.resource.prototype = {
        /**
         * Always fixed to 'uri' for resources.
         * @type String
         */
        type: 'uri',
        /**
         * The URI for the resource.
         * @type jQuery.rdf.uri
         */
        value: undefined,

        init: function (value, options) {
            var m, prefix, uri, opts;
            if (typeof value === 'string') {
                m = uriRegex.exec(value);
                opts = Utils.extend({}, rdf.resource.defaults, options);
                if (m !== null) {
                    this.value = URI.resolve(m[1].replace(/\\>/g, '>'), opts.base);
                } else if (value.substring(0, 1) === ':') {
                    uri = opts.namespaces[''];
                    if (uri === undefined) {
                        throw "Malformed Resource: No namespace binding for default namespace in " +
value;
                    } else {
                        this.value = URI.resolve(uri + value.substring(1));
                    }
                } else if (value.substring(value.length - 1) === ':') {
                    prefix = value.substring(0, value.length - 1);
                    uri = opts.namespaces[prefix];
                    if (uri === undefined) {
                        throw "Malformed Resource: No namespace binding for prefix " + prefix + " in " +
value;
                    } else {
                        this.value = URI.resolve(uri);
                    }
                } else {
                    try {
                        this.value = CURIE(value, { namespaces: opts.namespaces });
                    } catch (e) {
```

```
                        throw "Malformed Resource: Bad format for resource " + e;
                    }
                }
            } else {
                this.value = value;
            }
            return this;
        }, // end init

        /**
         * Returns a <a href="http://n2.talis.com/wiki/RDF_JSON_Specification">RDF/JSON</a>
representation of this triple.
         * @returns {Object}
         */
        dump: function () {
            return {
                type: 'uri',
                value: this.value.toString()
            };
        },

        /**
         * Returns a string representing this resource in Turtle format.
         * @returns {String}
         */
        toString: function () {
            return '<' + this.value + '>';
        }
    };

    rdf.resource.fn.init.prototype = rdf.resource.fn;

    rdf.resource.defaults = {
        base: URI.base(),
        namespaces: {}
    };

    /**
     * A {@link jQuery.rdf.resource} for rdf:type
     * @constant
     * @type jQuery.rdf.resource
     */
    rdf.type = rdf.resource('<' + rdfNs + 'type>');
    /**
     * A {@link jQuery.rdf.resource} for rdfs:label
     * @constant
     * @type jQuery.rdf.resource
     */
    rdf.label = rdf.resource('<' + rdfsNs + 'label>');
    /**
     * A {@link jQuery.rdf.resource} for rdf:first
     * @constant
     * @type jQuery.rdf.resource
     */
    rdf.first = rdf.resource('<' + rdfNs + 'first>');
    /**
     * A {@link jQuery.rdf.resource} for rdf:rest
     * @constant
     * @type jQuery.rdf.resource
     */
    rdf.rest = rdf.resource('<' + rdfNs + 'rest>');
    /**
     * A {@link jQuery.rdf.resource} for rdf:nil
     * @constant
     * @type jQuery.rdf.resource
     */
    rdf.nil = rdf.resource('<' + rdfNs + 'nil>');
```

```
    /**
     * A {@link jQuery.rdf.resource} for rdf:subject
     * @constant
     * @type jQuery.rdf.resource
     */
    rdf.subject = rdf.resource('<' + rdfNs + 'subject>');
    /**
     * A {@link jQuery.rdf.resource} for rdf:property
     * @constant
     * @type jQuery.rdf.resource
     */
    rdf.property = rdf.resource('<' + rdfNs + 'property>');
    /**
     * A {@link jQuery.rdf.resource} for rdf:object
     * @constant
     * @type jQuery.rdf.resource
     */
    rdf.object = rdf.resource('<' + rdfNs + 'object>');

    /**
     * <p>Creates a new jQuery.rdf.blank object. This should be invoked as a method rather than
constructed using new; indeed you will not usually want to generate these objects directly, since they
are automatically created from strings where necessary, such as by {@link jQuery.rdf#add}.</p>
     * @class Represents an RDF blank node.
     * @param {String} value A representation of the blank node in the format <code>_:<var>id</var></
code> or <code>[]</code> (which automatically creates a new blank node with a unique ID).
     * @returns {jQuery.rdf.blank} The newly-created blank node.
     * @throws {String} Errors if the string is not in a recognised format.
     * @example newBlank = rdf.blank('[]');
     * @example identifiedBlank = rdf.blank('_:fred');
     * @see jQuery.rdf.pattern
     * @see jQuery.rdf.triple
     * @see jQuery.rdf.resource
     * @see jQuery.rdf.literal
     */
    rdf.blank = function (value) {
        var blank;
        if (memBlank[value]) {
            return memBlank[value];
        }
        blank = new rdf.blank.fn.init(value);
        if (memBlank[blank]) {
            return memBlank[blank];
        } else {
            memBlank[blank] = blank;
            return blank;
        }
    };

    rdf.blank.fn = rdf.blank.prototype = {
        /**
         * Always fixed to 'bnode' for blank nodes.
         * @type String
         */
        type: 'bnode',
        /**
         * The value of the blank node in the format <code>_:<var>id</var></code>
         * @type String
         */
        value: undefined,
        /**
         * The id of the blank node.
         * @type String
         */
        id: undefined,

        init: function (value) {
```

```
            if (value === '[]') {
                this.id = blankNodeID();
                this.value = '_:' + this.id;
            } else if (value.substring(0, 2) === '_:') {
                this.id = value.substring(2);
                this.value = value;
            } else {
                throw "Malformed Blank Node: " + value + " is not a legal format for a blank node";
            }
            return this;
        },

        /**
         * Returns a <a href="http://n2.talis.com/wiki/RDF_JSON_Specification">RDF/JSON</a>
representation of this blank node.
         * @returns {Object}
         */
        dump: function () {
            return {
                type: 'bnode',
                value: this.value
            };
        },

        /**
         * Returns the value this blank node.
         * @returns {String}
         */
        toString: function () {
            return this.value;
        }
    };

    rdf.blank.fn.init.prototype = rdf.blank.fn;

    /**
     * <p>Creates a new jQuery.rdf.literal object. This should be invoked as a method rather than
constructed using new; indeed you will not usually want to generate these objects directly, since they
are automatically created from strings where necessary, such as by {@link jQuery.rdf#add}.</p>
     * @class Represents an RDF literal.
     * @param {String|boolean|Number} value Either the value of the literal or a string representation of
it. If the datatype or lang options are specified, the value is taken as given. Otherwise, if it's a
Javascript boolean or numeric value, it is interpreted as a value with a xsd:boolean or xsd:double
datatype. In all other cases it's interpreted as a literal as defined in <a href="http://www.w3.org/
TeamSubmission/turtle/#literal">Turtle</a> syntax</a>.
     * @param {Object} [options] Initialisation options for the literal.
     * @param {String} [options.datatype] The datatype for the literal. This should be a safe CURIE; in
other words, it can be in the format <code><var>uri</var></code> or <code>[<var>curie</var>]</code>. Must
not be specified if options.lang is also specified.
     * @param {String} [options.lang] The language for the literal. Must not be specified if
options.datatype is also specified.
     * @param {Object} [options.namespaces] An object representing a set of namespace bindings used when
interpreting a CURIE in the datatype.
     * @param {String|jQuery.uri} [options.base] The base URI used to interpret a relative URI in the
datatype.
     * @returns {jQuery.rdf.literal} The newly-created literal.
     * @throws {String} Errors if the string is not in a recognised format or if both options.datatype
and options.lang are specified.
     * @example trueLiteral = rdf.literal(true);
     * @example numericLiteral = rdf.literal(5);
     * @example dateLiteral = rdf.literal('"2009-07-13"^^xsd:date', { namespaces: ns });
     * @see jQuery.rdf.pattern
     * @see jQuery.rdf.triple
     * @see jQuery.rdf.resource
     * @see jQuery.rdf.blank
     */
    rdf.literal = function (value, options) {
```

```
        var literal;
        if (memLiteral[value]) {
            return memLiteral[value];
        }
        literal = new rdf.literal.fn.init(value, options);
        if (memLiteral[literal]) {
            return memLiteral[literal];
        } else {
            memLiteral[literal] = literal;
            return literal;
        }
    };

    rdf.literal.fn = rdf.literal.prototype = {
        /**
         * Always fixed to 'literal' for literals.
         * @type String
         */
        type: 'literal',
        /**
         * The value of the literal as a string.
         * @type String
         */
        value: undefined,
        /**
         * The language of the literal, if it has one; otherwise undefined.
         * @type String
         */
        lang: undefined,
        /**
         * The datatype of the literal, if it has one; otherwise undefined.
         * @type jQuery.uri
         */
        datatype: undefined,

        init: function (value, options) {
            var m,
                datatype,
                opts = Utils.extend({}, rdf.literal.defaults, options);
            if (opts.datatype) {
                datatype = CURIE.safeCurie(opts.datatype, { namespaces: opts.namespaces });
            }
            if (opts.lang !== undefined && opts.datatype !== undefined && datatype.toString() !== (rdfNs
+ 'XMLLiteral')) {
                throw "Malformed Literal: Cannot define both a language and a datatype for a literal (" +
value + ")";
            }
            if (opts.datatype !== undefined) {
                datatype = CURIE.safeCurie(opts.datatype, { namespaces: opts.namespaces });
                Utils.extend(this, TypedValue(value.toString(), datatype));
                if (datatype.toString() === rdfNs + 'XMLLiteral') {
                    this.lang = opts.lang;
                }
            } else if (opts.lang !== undefined) {
                this.value = value.toString();
                this.lang = opts.lang;
            } else if (typeof value === 'boolean') {
                Utils.extend(this, TypedValue(value.toString(), xsdNs + 'boolean'));
            } else if (typeof value === 'number') {
                Utils.extend(this, TypedValue(value.toString(), xsdNs + 'double'));
            } else if (value === 'true' || value === 'false') {
                Utils.extend(this, TypedValue(value, xsdNs + 'boolean'));
            } else if (TypedValue.valid(value, xsdNs + 'integer')) {
                Utils.extend(this, TypedValue(value, xsdNs + 'integer'));
            } else if (TypedValue.valid(value, xsdNs + 'decimal')) {
                Utils.extend(this, TypedValue(value, xsdNs + 'decimal'));
            } else if (TypedValue.valid(value, xsdNs + 'double') &&
```

```
                !/^\s*([\-\+]?INF|NaN)\s*$/.test(value)) {  // INF, -INF and NaN aren't valid literals in
Turtle
                Utils.extend(this, TypedValue(value, xsdNs + 'double'));
            } else {
                m = literalRegex.exec(value);
                if (m !== null) {
                    this.value = (m[2] || m[4]).replace(/\\"/g, '"').replace(/\\n/g, '\n').replace(/\\t/
g, '\t').replace(/\\r/g, '\r');
                    if (m[9]) {
                        datatype = rdf.resource(m[9], opts);
                        Utils.extend(this, TypedValue(this.value, datatype.value));
                    } else if (m[7]) {
                        this.lang = m[7];
                    }
                } else {
                    throw "Malformed Literal: Couldn't recognise the value " + value;
                }
            }
            return this;
        }, // end init

        /**
         * Returns a <a href="http://n2.talis.com/wiki/RDF_JSON_Specification">RDF/JSON</a>
representation of this blank node.
         * @returns {Object}
         */
        dump: function () {
            var e = {
                type: 'literal',
                value: this.value.toString()
            };
            if (this.lang !== undefined) {
                e.lang = this.lang;
            } else if (this.datatype !== undefined) {
                e.datatype = this.datatype.toString();
            }
            return e;
        },

        /**
         * Returns a string representing this resource in <a href="http://www.w3.org/TeamSubmission/
turtle/#literal">Turtle format</a>.
         * @returns {String}
         */
        toString: function () {
            var val = '"' + this.value + '"';
            if (this.lang !== undefined) {
                val += '@' + this.lang;
            } else if (this.datatype !== undefined) {
                val += '^^<' + this.datatype + '>';
            }
            return val;
        }
    };

    rdf.literal.fn.init.prototype = rdf.literal.fn;

    rdf.literal.defaults = {
        base: URI.base(),
        namespaces: {},
        datatype: undefined,
        lang: undefined
    };
    return rdf;
});define([
    "./../graphite/utils"
], function (Utils) {
```

```
    /*
     * $ URIs @VERSION
     *
     * Copyright (c) 2008,2009 Jeni Tennison
     * Licensed under the MIT (MIT-LICENSE.txt)
     *
     */
    /**
     * @fileOverview $ URIs
     * @author <a href="mailto:jeni@jenitennison.com">Jeni Tennison</a>
     * @copyright (c) 2008,2009 Jeni Tennison
     * @license MIT license (MIT-LICENSE.txt)
     * @version 1.0
     */
    /**
     * @class
     * @name jQuery
     * @exports $ as jQuery
     * @description rdfQuery is a <a href="http://jquery.com/">jQuery</a> plugin. The only fields and
methods listed here are those that come as part of the rdfQuery library.
     */
    var global = typeof window === 'undefined' ? this : window;

    var mem = {},
        uriRegex = /^(([a-z][\-a-z0-9+\.]*):)?(\/\/([^\/?#]+))?([^?#]*)?(\?([^#]*))?(#(.*))?$/i,
        parseURI = function (u) {
            var m = u.match(uriRegex);
            if (m === null) {
                throw "Malformed URI: " + u;
            }
            //console.log("PARSED URI, the match", m);
            return {
                scheme: m[1] ? m[2].toLowerCase() : undefined,
                authority: m[3] ? m[4] : undefined,
                path: m[5] || '',
                query: m[6] ? m[7] : undefined,
                fragment: m[8] ? m[9] : undefined
            };
        },
        removeDotSegments = function (u) {
            var r = '', m = [];
            if (/\./.test(u)) {
                while (u !== undefined && u !== '') {
                    if (u === '.' || u === '..') {
                        u = '';
                    } else if (/^\.\.\//.test(u)) { // starts with ../
                        u = u.substring(3);
                    } else if (/^\.\//.test(u)) { // starts with ./
                        u = u.substring(2);
                    } else if (/^\/\.(\/|$)/.test(u)) { // starts with /./ or consists of /.
                        u = '/' + u.substring(3);
                    } else if (/^\/\.\.(\/|$)/.test(u)) { // starts with /../ or consists of /..
                        u = '/' + u.substring(4);
                        r = r.replace(/\/?[^\/]+$/, '');
                    } else {
                        m = u.match(/^(\/?[^\/]*)(\/.*)?$/);
                        u = m[2];
                        r = r + m[1];
                    }
                }
                return r;
            } else {
                return u;
            }
        },
        merge = function (b, r) {
            if (b.authority !== '' && (b.path === undefined || b.path === '')) {
```

```
                return '/' + r;
            } else {
                return b.path.replace(/[^\/]+$/, '') + r;
            }
        },
        uri;
    /**
     * Creates a new jQuery.uri object. This should be invoked as a method rather than constructed using
new.
     * @class Represents a URI
     * @param {String} [relative='']
     * @param {String|jQuery.uri} [base] Defaults to the base URI of the page
     * @returns {jQuery.uri} The new jQuery.uri object.
     * @example uri = jQuery.uri('/my/file.html');
     */
    uri = function (relative, base) {
        var u;
        relative = relative || '';
        if (mem[relative]) {
            return mem[relative];
        }
        base = base || uri.base();
        if (typeof base === 'string') {
            base = uri.absolute(base);
        }
        u = new uri.prototype.init(relative, base);
        if (mem[u]) {
            return mem[u];
        } else {
            mem[u] = u;
            return u;
        }
    };
    uri.prototype = {
        /**
         * The scheme used in the URI
         * @type String
         */
        scheme: undefined,
        /**
         * The authority used in the URI
         * @type String
         */
        authority: undefined,
        /**
         * The path used in the URI
         * @type String
         */
        path: undefined,
        /**
         * The query part of the URI
         * @type String
         */
        query: undefined,
        /**
         * The fragment part of the URI
         * @type String
         */
        fragment: undefined,
        init: function (relative, base) {
            //console.log("IN URI", relative, base);
            var r = {};
            base = base || {};
            Utils.extend(this, parseURI(relative));
            //console.log("URI PARSED");
            if (this.scheme === undefined) {
                this.scheme = base.scheme;
```

```
            if (this.authority !== undefined) {
                this.path = removeDotSegments(this.path);
            } else {
                this.authority = base.authority;
                if (this.path === '') {
                    this.path = base.path;
                    if (this.query === undefined) {
                        this.query = base.query;
                    }
                } else {
                    if (!/^\//.test(this.path)) {
                        this.path = merge(base, this.path);
                    }
                    this.path = removeDotSegments(this.path);
                }
            }
        }
        if (this.scheme === undefined) {
            throw "Malformed URI: URI is not an absolute URI and no base supplied: " + relative;
        }
        return this;
    },
    /**
     * Resolves a relative URI relative to this URI
     * @param {String} relative
     * @returns jQuery.uri
     */
    resolve: function (relative) {
        return uri(relative, this);
    },
    /**
     * Creates a relative URI giving the path from this URI to the absolute URI passed as a parameter
     * @param {String|jQuery.uri} absolute
     * @returns String
     */
    relative: function (absolute) {
        var aPath, bPath, i = 0, j, resultPath = [], result = '';
        if (typeof absolute === 'string') {
            absolute = uri(absolute, {});
        }
        if (absolute.scheme !== this.scheme ||
            absolute.authority !== this.authority) {
            return absolute.toString();
        }
        if (absolute.path !== this.path) {
            aPath = absolute.path.split('/');
            bPath = this.path.split('/');
            if (aPath[1] !== bPath[1]) {
                result = absolute.path;
            } else {
                while (aPath[i] === bPath[i]) {
                    i += 1;
                }
                j = i;
                for (; i < bPath.length - 1; i += 1) {
                    resultPath.push('..');
                }
                for (; j < aPath.length; j += 1) {
                    resultPath.push(aPath[j]);
                }
                result = resultPath.join('/');
            }
            result = absolute.query === undefined ? result : result + '?' + absolute.query;
            result = absolute.fragment === undefined ? result : result + '#' + absolute.fragment;
            return result;
        }
        if (absolute.query !== undefined && absolute.query !== this.query) {
```

```
                return '?' + absolute.query + (absolute.fragment === undefined ? '' : '#' +
absolute.fragment);
            }
            if (absolute.fragment !== undefined && absolute.fragment !== this.fragment) {
                return '#' + absolute.fragment;
            }
            return '';
        },
        /**
         * Returns the URI as an absolute string
         * @returns String
         */
        toString: function () {
            var result = '';
            if (this._string) {
                return this._string;
            } else {
                result = this.scheme === undefined ? result : (result + this.scheme + ':');
                result = this.authority === undefined ? result : (result + '//' + this.authority);
                result = result + this.path;
                result = this.query === undefined ? result : (result + '?' + this.query);
                result = this.fragment === undefined ? result : (result + '#' + this.fragment);
                this._string = result;
                return result;
            }
        }
    };
    uri.prototype.init.prototype = uri.prototype;
    /**
     * Creates a {@link jQuery.uri} from a known-to-be-absolute URI
     * @param {String}
         * @returns {jQuery.uri}
     */
    uri.absolute = function (u) {
        return uri(u, {});
    };
    /**
     * Returns the base URI of the page
     * TODO: This must be handled in a correct manner
     * @returns {jQuery.uri}
     */
    uri.base = function () {
        return global.document && typeof global.document !== undefined
            ? global.document.location.href
            : "http://example.com/";
    };
    /**
     * Creates a {@link jQuery.uri} from a relative URI and an optional base URI
     * @returns {jQuery.uri}
     * @see jQuery.uri
     */
    uri.resolve = function (relative, base) {
        return uri(relative, base);
    };
    /**
     * Creates a string giving the relative path from a base URI to an absolute URI
     * @param {String} absolute
     * @param {String} base
     * @returns {String}
     */
    uri.relative = function (absolute, base) {
        return uri(base, {}).relative(absolute);
    };
    return uri;
});if (typeof module === "object" && typeof require === "function") {
    var moduleHttp = require("http");
    var moduleUrl = require("url");
```

```
}

define(["../utils"], function (Utils) {
    var http = function (options) {
        return new http.prototype.init(options);
    };
    http.prototype = {
        init: function (options) {
            this.options = getOptions(options);
            if(this.options.success) {
                send.call(this, this.options.success);
            }
        },
        send: send
    }
    http.prototype.init.prototype = http.prototype;

    function send (callback) {
        var obj = this,
            status,
            headers,
            result = "",
            req = moduleHttp.request(obj.options, function (res) {
                status = res.statusCode;
                headers = res.headers;
                if(obj.options.followRedirect && headers.location) {
                    obj.options = getOptions({
                        "uri": headers.location
                    });
                    return obj.send(callback);
                }
                res.setEncoding('utf8');
                res.on('data', function (chunk) {
                    result += chunk;
                });
                res.on('end', function () {
                    callback(null, result, status, res);
                });
            });
        req.on('error', function (e) {
            callback(e.message);
        });
        req.end();
    };

    function getOptions(options) {
        var opts = {
                followRedirect: false
            },
            uri = Utils.extract(options, "uri");
        if(uri && Utils.isUri(uri)) {
            Utils.extend(opts, moduleUrl.parse(uri));
        } else if(uri) {
            throw "Not a valid uri given to loader";
        }
        return Utils.extend(opts, options);
    }
    return http;
});define(["./xhr", "../utils"], function (XHR, Utils) {
    "use strict";
    var proxy = function (options) {
        return new proxy.prototype.init(options);
    };
    proxy.prototype = {
        init: function (options) {
            var proxyUri = Utils.extract(options, "proxy"),
                parts = Utils.parseUri(options.uri),
```

```
                success = Utils.extract(options, "success"),
                uri = proxyUri + "?host=" + parts.host;
            if (parts.userInfo !== "") {
                uri += "&auth=" + parts.userInfo;
            }
            if (parts.relative !== "") {
                uri += "&path=" + parts.relative;
            }
            if (parts.port !== "") {
                uri += "&port=" + parts.port;
            }
            options.uri = uri;
            this.xhr = XHR(options);
            if (success) {
                this.send(success);
            }
        },
        abort: function () {
            this.xhr.abort();
        },
        send: function (callback) {
            this.xhr.send(callback);
        }
    };
    proxy.prototype.init.prototype = proxy.prototype;
    return proxy;
});define([
    "../utils"
], function (Utils) {
    "use strict";
    var xhr = function XHR(options) {
        return new xhr.prototype.init(options);
    };
    xhr.prototype = {
        init: function (options) {
            this.options = Utils.extend({
                "asynchronous": true,
                "method": "GET",
                "uri": window.location
            }, options);
            this.req = new XMLHttpRequest();
            if (this.options.success) {
                this.send(this.options.success);
            }
        },
        abort: function () {
            this.req.abort();
        },
        send: function (callback) {
            //console.log("IN XHR, SENDING", this.options);
            var that = this;
            this.req.open(this.options.method, this.options.uri, this.options.asynchronous);
            this.req.onerror = function () {
                //console.log("IN XHR, ERROR");
                callback("There was an error making the request");
            };
            this.req.onload = function () {
                //console.log("IN XHR, SUCCESS");
                callback(null, that.req.responseText, that.req.status, that.req);
            };
            this.req.ontimeout = function () {
                //console.log("IN XHR, TIMEOUT");
                callback("The request timed out");
            };
            this.req.send();
        }
    };
```

```
    xhr.prototype.init.prototype = xhr.prototype;
    return xhr;
});/*global define */
define([
    "../../rdfstore/sparql-parser/sparql_parser",
    "../utils"
], function (SparqlParser, Utils) {
    "use strict";
    var aliasRegex = /^\(/,
        ascRegex = /^(ASC|asc)/,
        asRegex = /^(AS|as)/,
        asteriskRegex = /^\*/,
        avgRegex = /^(AVG|avg)/,
        baseRegex = /^(BASE|base)/,
        bnodeRegex = /^(BNODE|bnode)/,
        colonRegex = /^:/,
        commaRegex = /^,/,
        countRegex = /^(COUNT|count)/,
        curieRegex = /^[a-zA-Z0-9]+:[a-zA-Z0-9]+/,
        curlyBracketLeftRegex = /^\{/,
        curlyBracketRightRegex = /^\}/,
        datatypeRegex = /^\^\^/,
        descRegex = /^(DESC|desc)/,
        dotRegex = /^\./,
        equalsRegex = /^=/,
        filterRegex = /^(FILTER|filter)/,
        greaterOrEqualsRegex = /^>=/,
        greaterRegex = /^>/,
        langRegex = /^@/,
        lesserOrEqualsRegex = /^<=/,
        lesserRegex = /^</,
        literalRegex = /^"/,
        literalStringRegex = /^[a-zA-Z0-9\s]*/,
        maxRegex = /^(MAX|max)/,
        minRegex = /^(MIN|min)/,
        notEqualsRegex = /^!=/,
        numericRegex = /^[0-9]+/,
        optionalRegex = /^(OPTIONAL|optional)/,
        parenthesisLeft = /^\(/,
        parenthesisRight = /^\)/,
        prefixRegex = /^(PREFIX|prefix)/,
        regexRegex = /^(REGEX|regex)/,
        semicolonRegex = /^;/,
        stringRegex = /^[a-zA-Z0-9]*/,
        sumRegex = /^(SUM|sum)/,
        typeRegex = /^a/,
        uriRegex = /^http:\/\/[a-zA-Z0-9#_\-.\/]+/,
        variableRegex = /^\?/,
        whereRegex = /^(WHERE|where)/,
        wsRegex = /^(\u0009|\u000A|\u000D|\u0020|#([\u000A\u000D])*)+/,
        token = {
            base: function (value) {
                return {
                    "token": "base",
                    "value": value
                };
            },
            expression: function (expressionType, primaryExpression, value) {
                return {
                    "expressionType": expressionType,
                    "primaryexpression": primaryExpression,
                    "token": "expression",
                    "value": value
                };
            },
            literal: function (value, lang, type) {
                return {
```

```
                "lang": lang,
                "token": "literal",
                "type": type,
                "value": value
            };
        },
        optional: function (value) {
            return {
                "token": "optionalgraphpattern",
                "value": value
            };
        },
        prefix: function (prefix, local) {
            return {
                "local": local,
                "prefix": prefix,
                "token": "prefix"
            };
        },
        uri: function (options) {
            return {
                "prefix": options.prefix || null,
                "suffix": options.suffix || null,
                "token": "uri",
                "value": options.value || null
            };
        },
        "var": function (value) {
            return {
                "token": "var",
                "value": value
            };
        },
        /**
         *
         * @param kind
         * @param [value]
         * @return {Object}
         */
        "variable": function (kind, value) {
            var tmp = {
                "kind": kind,
                "token": "variable"
            };
            if (value) {
                tmp.value = value;
            }
            return tmp;
        }
    };
function expect(data, regex) {
    if (!regex.test(data)) {
        throw new Error("Didn't meet expected token: " + data);
    }
    return data.substr(regex.exec(data)[0].length);
}
/**
 *
 * @param tokenizer
 * @param data
 * @param [options]
 * @return {Object}
 */
function bpgPart(tokenizer, data, options) {
    //console.log("BPGPART", data);
    var lToken, part;
    if (typeRegex.test(data)) {
```

```
            data = expect(data, typeRegex);
            lToken = token.uri({
                value: "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
            });
        } else if (variableRegex.test(data)) {
            part = tokenizer["var"](data.substr(1));
            lToken = part["var"];
            data = part.remainder;
        } else if (literalRegex.test(data)) {
            part = tokenizer.literal(data, options);
            lToken = part.literal;
            data = part.remainder;
        } else if (curieRegex.test(data) || lesserRegex.test(data)) {
            part = tokenizer.uri(data, options);
            lToken = part.uri;
            data = part.remainder;
        } else {
            part = tokenizer.literal(data, options);
            lToken = part.literal;
            data = part.remainder;
        }
        return {
            "remainder": data,
            "token": lToken
        };
    }
    function expressionAggregate(tokenizer, type, data) {
        var remainder = expect(data.substr(type.length), parenthesisLeft),
            expression = tokenizer.expression(remainder);
        remainder = expect(expression.remainder, parenthesisRight);
        return {
            "expression": {
                "aggregateType": type,
                "distinct": "",
                "expression": expression.expression,
                "expressionType": "aggregate",
                "token": "expression"
            },
            "remainder": remainder
        };
    }
    /**
     *
     * @param data
     * @param [options]
     * @return {Object}
     */
    function expressionLiteral(tokenizer, data, options) {
        var literal = tokenizer.literal(data, options);
        return {
            "expression": {
                "expressionType": "atomic",
                "primaryexpression": "rdfliteral",
                "token": "expression",
                "value": literal.literal
            },
            "remainder": literal.remainder
        };
    }
    function expressionNumericLiteral(tokenizer, data, options) {
        var literal = tokenizer.literal(data, options);
        return {
            "expression": {
                "expressionType": "atomic",
                "primaryexpression": "numericliteral",
                "token": "expression",
                "value": literal.literal
```

```
            },
            "remainder": literal.remainder
        };
    }
    function ws(data, opts) {
        opts = opts || {};
        if ((opts.required || false) && !wsRegex.test(data)) {
            throw ("Invalid sparql: Required whitespace is missing!");
        }
        //console.log("IN SPARQL, WS", data);
        return data.replace(wsRegex, '');
    }
    function expressionRegex(tokenizer, data) {
        var flags, pattern, text;
        data = expect(data, regexRegex);
        data = expect(data, parenthesisLeft);
        text = tokenizer.expression(data);
        data = text.remainder;
        text = text.expression;
        data = expect(data, commaRegex);
        data = ws(data);
        pattern = tokenizer.expression(data);
        data = pattern.remainder;
        pattern = pattern.expression;
        if (commaRegex.test(data)) {
            data = expect(data, commaRegex);
            data = ws(data);
            flags = tokenizer.expression(data);
            data = flags.remainder;
            flags = flags.expression;
        }
        data = expect(data, parenthesisRight);
        return {
            "expression": {
                "expressionType": "regex",
                "flags": flags,
                "pattern": pattern,
                "text": text,
                "token": "expression"
            },
            "remainder": data
        };
    }
    function expressionRelationalOp2(tokenizer, data, regex) {
        var op2;
        data = expect(data, regex);
        data = ws(data, { required: true });
        op2 = tokenizer.expression(data);
        data = ws(op2.remainder);
        op2 = op2.expression;
        return {
            "op2": op2,
            "remainder": data
        };
    }
    function expressionRelational(tokenizer, data, op1) {
        var op2,
            expression = {
                "expressionType": "relationalexpression",
                "token": "expression"
            };
        if (!op1) {
            if (variableRegex.test(data)) {
                data = expect(data, variableRegex);
                op1 = tokenizer.expression(data);
                data = ws(op1.remainder, { required: true });
                op1 = op1["var"];
```

```
            } else {
                throw new Error("Expressional relation sign not recognized: " + data);
            }
        }
        if (equalsRegex.test(data)) {
            op2 = expressionRelationalOp2(tokenizer, data, equalsRegex);
            data = op2.remainder;
            op2 = op2.op2;
            expression.operator = "=";
        } else if (greaterOrEqualsRegex.test(data)) {
            op2 = expressionRelationalOp2(tokenizer, data, greaterOrEqualsRegex);
            data = op2.remainder;
            op2 = op2.op2;
            expression.operator = ">=";
        } else if (greaterRegex.test(data)) {
            op2 = expressionRelationalOp2(tokenizer, data, greaterRegex);
            data = op2.remainder;
            op2 = op2.op2;
            expression.operator = ">";
        } else if (lesserOrEqualsRegex.test(data)) {
            op2 = expressionRelationalOp2(tokenizer, data, lesserOrEqualsRegex);
            data = op2.remainder;
            op2 = op2.op2;
            expression.operator = "<=";
        } else if (lesserRegex.test(data)) {
            op2 = expressionRelationalOp2(tokenizer, data, lesserRegex);
            data = op2.remainder;
            op2 = op2.op2;
            expression.operator = "<";
        } else if (notEqualsRegex.test(data)) {
            op2 = expressionRelationalOp2(tokenizer, data, notEqualsRegex);
            data = op2.remainder;
            op2 = op2.op2;
            expression.operator = "!=";
        } else {
            throw new Error("Expressional relation sign not recognized: " + data);
        }
        expression.op1 = op1;
        expression.op2 = op2;
        return {
            "expression": expression,
            "remainder": data
        };
    }
    function whereBGP(triplesContext, pattern, options) {
        switch (pattern.kind) {
        case "BGP":
            //console.log("IN SPARQL, WHERE BGP BGP");
            return {
                "kind": "BGP",
                "value": pattern.value.concat(triplesContext)
            };
        case "EMPTY_PATTERN":
            //console.log("IN SPARQL, WHERE BGP EMPTY_PATTERN");
            return {
                "kind": "BGP",
                "value": triplesContext
            };
        case "FILTER":
            //console.log("IN SPARQL, WHERE BGP FILTER");
            pattern.value = whereBGP(triplesContext, pattern.value, options);
            return pattern;
        case "JOIN":
            //console.log("IN SPARQL, WHERE BGP JOIN");
            pattern.rvalue.value = pattern.rvalue.value.concat(triplesContext);
            return pattern;
        case "LEFT_JOIN":
```

```
            //console.log("IN SPARQL, WHERE BGP LEFT_JOIN");
            return {
                "kind": "JOIN",
                "lvalue": pattern,
                "rvalue": {
                    "kind": "BGP",
                    "value": triplesContext
                }
            };
        default:
            return {
                "kind": "BGP",
                "value": triplesContext
            };
        }
    }
    function whereFilter(token, pattern) {
        token = {
            "filter": true,
            "kind": "LEFT_JOIN",
            "lvalue": pattern,
            "rvalue": {
                "kind": "BGP",
                "value": token.optional.value.patterns[0].triplesContext
            }
        };
        return token;
    }
    return {
        "alias": function (data) {
            var alias;
            if (!asRegex.test(data)) {
                throw new Error("Couldn't parse alias: " + data);
            }
            data = ws(data.substr(2), { required: true });
            if (variableRegex.test(data)) {
                alias = this["var"](data.substr(1));
                return {
                    "alias": alias["var"],
                    "remainder": alias.remainder
                };
            } else {
                throw new Error("Alias don't know how to parse the following " + data);
            }
        },
        base: function (data) {
            //console.log("In tokenizer (SPARQL)", data);
            var value = uriRegex.exec(data)[0];
            return {
                "base": token.base(value),
                "remainder": data.substr(value.length)
            };
        },
        /**
         *
         * @param data
         * @param triplesContext
         * @param [options]
         * @return {Object}
         */
        basicgraphpattern: function (data, triplesContext, options) {
            options = options || {};
            //console.log("STARTING BGP", data, subject);
            //console.log("IN SPARQL, BGP", data);
            //console.log(options);
            var subject = !options.subject ? bpgPart(this, data, options) : {
                    "remainder": data,
```

```
                    "token": options.subject
                },
                predicate = !options.predicate ? bpgPart(this, ws(subject.remainder, { required: true }),
options) : {
                    "remainder": data,
                    "token": options.predicate
                },
                object = bpgPart(this, ws(predicate.remainder, { required: true }), options),
                remainder = ws(object.remainder),
                value = {
                    "object": object.token,
                    "predicate": predicate.token,
                    "subject": subject.token
                };
            if (options.variables) {
                value.variables = options.variables;
            }
            triplesContext = triplesContext || [];
            triplesContext.push(value);
            //console.log("IN SPARQL, BGP, parsed triple", triplesContext);
            if (dotRegex.test(remainder)) {
                remainder = ws(remainder.substr(1));
            } else if (semicolonRegex.test(remainder)) {
                remainder = expect(remainder, semicolonRegex);
                options.subject = subject.token;
                return this.basicgraphpattern(remainder, triplesContext, options);
            } else if (commaRegex.test(remainder)) {
                remainder = expect(remainder, commaRegex);
                options.subject = subject.token;
                options.predicate = predicate.token;
                return this.basicgraphpattern(remainder, triplesContext, options);
            }
            if (variableRegex.test(remainder)) {
                return this.basicgraphpattern(remainder, triplesContext, options);
            }
            if (options.basicgraphpattern && options.modifiedPattern) {
                triplesContext = options.basicgraphpattern.triplesContext.concat(triplesContext);
            }
            return {
                "basicgraphpattern": {
                    "token": "basicgraphpattern",
                    "triplesContext": triplesContext
                },
                "remainder": remainder
            };
        },
        /**
         *
         * @param data
         * @param [options]
         * @return {Object}
         */
        expression: function (data, options) {
            var value,
                arg,
                expression,
                remainder,
                irireforfunction;
            if (variableRegex.test(data)) {
                value = this["var"](data.substr(1));
                expression =  {
                    "expressionType": "atomic",
                    "primaryexpression": "var",
                    "token": "expression",
                    "value": value["var"]
                };
                data = ws(value.remainder);
```

```
            if (equalsRegex.test(data) ||
                    greaterOrEqualsRegex.test(data) ||
                    greaterRegex.test(data) ||
                    lesserOrEqualsRegex.test(data) ||
                    lesserRegex.test(data) ||
                    notEqualsRegex.test(data)) {
                return expressionRelational(this, data, expression);
            }
            return {
                "expression": expression,
                "remainder": data
            };
        } else if (lesserRegex.test(data)) {
            irireforfunction = this.uri(data, options);
            return {
                "expression": {
                    "args": undefined,
                    "expressionType": "irireforfunction",
                    "iriref": irireforfunction.uri,
                    "token": "expression"
                },
                "remainder": irireforfunction.remainder
            };
        } else if (literalRegex.test(data)) {
            return expressionLiteral(this, data, options);
        } else if (bnodeRegex.test(data)) {
            remainder = expect(data.substr(5), parenthesisLeft);
            arg = this.expression(remainder);
            remainder = expect(arg.remainder, parenthesisRight);
            return {
                "expression": {
                    "args": [ arg.expression ],
                    "builtincall": "bnode",
                    "expressionType": "builtincall",
                    "token": "expression"
                },
                "remainder": remainder
            };
        } else if (avgRegex.test(data)) {
            return expressionAggregate(this, "avg", data);
        } else if (countRegex.test(data)) {
            return expressionAggregate(this, "count", data);
        } else if (maxRegex.test(data)) {
            return expressionAggregate(this, "max", data);
        } else if (minRegex.test(data)) {
            return expressionAggregate(this, "min", data);
        } else if (sumRegex.test(data)) {
            return expressionAggregate(this, "sum", data);
        } else if (regexRegex.test(data)) {
            return expressionRegex(this, data);
        } else if (numericRegex.test(data)) {
            return expressionNumericLiteral(this, data, options);
        } else if (stringRegex.test(data)) {
            return expressionLiteral(this, data, options);
        } else {
            throw new Error("Can't parse expression: " + data);
        }
    },
    /**
     *
     * @param data
     * @param [options]
     * @return {Object}
     */
    filter: function (data, options) {
        var value;
        data = expect(data, filterRegex);
```

```
            data = ws(data);
            if (parenthesisLeft.test(data)) {
                data = expect(data, parenthesisLeft);
                data = ws(data);
                value = this.expression(data, options);
                data = expect(value.remainder, parenthesisRight);
                data = ws(data);
            } else {
                value = this.expression(data, options);
                data = value.remainder;
            }
            return {
                "filter": {
                    "token": "filter",
                    "value": value.expression
                },
                "remainder": data
            };
        },
        group: function (data, group) {
            if (!group) {
                if (!variableRegex.test(data)) {
                    throw new Error("Not valid group-syntax: " + data);
                }
                group = [];
            }
            var variable;
            if (variableRegex.test(data)) {
                data = expect(data, variableRegex);
                variable = this["var"](data);
                group.push(variable["var"]);
                data = ws(variable.remainder);
                return this.group(data, group);
            }
            return {
                "group": group,
                "remainder": data
            };
        },
        /**
         *
         * @param data
         * @param filters
         * @param patterns
         * @param bgpindex
         * @param [options]
         * @return {Object}
         */
        groupgraphpattern: function (data, filters, patterns, bgpindex, options) {
            //console.log("IN SPARQL TOKENIZER, groupgraphpattern", patterns, bgpindex);
            options = options || {};
            var filter,
                pattern,
                triplesContext = [];
            if (variableRegex.test(data) || lesserRegex.test(data)) {
                if (Utils.isNumber(bgpindex)) {
                    triplesContext = patterns[bgpindex].triplesContext;
                    //console.log("IN SPARQL, GGP", triplesContext);
                } else {
                    bgpindex = patterns.length;
                }
                pattern = this.basicgraphpattern(data, triplesContext, options);
                patterns[bgpindex] = pattern.basicgraphpattern;
                data = ws(pattern.remainder);
                return this.groupgraphpattern(data, filters, patterns, bgpindex, options);
            } else if (filterRegex.test(data)) {
                filter = this.filter(data, options);
```

```
            data = ws(filter.remainder);
            filters.push(filter.filter);
            return this.groupgraphpattern(data, filters, patterns, null, options);
        } else if (optionalRegex.test(data)) {
            pattern = this.optional(data, options);
            patterns.push(pattern.optional);
            data = pattern.remainder;
            data = ws(data);
            return this.groupgraphpattern(data, filters, patterns, bgpindex, options);
        }
        return {
            "bgpindex": bgpindex,
            "groupgraphpattern": {
                "filters": filters,
                "patterns": patterns,
                "token": "groupgraphpattern"
            },
            "remainder": data
        };
    },
    /**
     *
     * @param data
     * @param [options]
     * @return {Object}
     */
    "literal": function (data, options) {
        var lang = null,
            type = null,
            value;
        if (literalRegex.test(data)) {
            data = data.substr(1);
            value = literalStringRegex.exec(data)[0];
            data = data.substr(value.length);
            data = expect(data, literalRegex);
            if (langRegex.test(data)) {
                data = data.substr(1);
                lang = stringRegex.exec(data)[0];
                data = data.substr(lang.length);
            } else if (datatypeRegex.test(data)) {
                data = data.substr(2);
                type = this.uri(data, options);
                data = type.remainder;
                type = type.uri;
            }
        } else {
            value = stringRegex.exec(data)[0];
            data = data.substr(value.length);
            if (Utils.isInteger(value)) {
                type = "http://www.w3.org/2001/XMLSchema#integer";
            }
        }
        return {
            "literal": {
                "lang": lang,
                "token": "literal",
                "type": type,
                "value": value
            },
            "remainder": data
        };
    },
    /**
     *
     * @param data
     * @param [options]
     * @return {Object}
```

```
     */
    "optional": function (data, options) {
        var groupgraphpattern;
        data = expect(data, optionalRegex);
        data = ws(data);
        data = expect(data, curlyBracketLeftRegex);
        data = ws(data);
        groupgraphpattern = this.groupgraphpattern(data, [], [], null, options);
        data = expect(groupgraphpattern.remainder, curlyBracketRightRegex);
        data = ws(data);
        return {
            "optional": token.optional(groupgraphpattern.groupgraphpattern),
            "remainder": data
        };
    },
    "order": function (data, order) {
        if (!order) {
            if (!variableRegex.test(data) && !ascRegex.test(data) && !descRegex.test(data)) {
                throw new Error("Not valid order-syntax: " + data);
            }
            order = [];
        }
        var expression;
        if (ascRegex.test(data)) {
            data = expect(data, ascRegex);
            data = expect(data, parenthesisLeft);
            expression = this.expression(data);
            data = expect(expression.remainder, parenthesisRight);
            data = ws(data);
            order.push({
                "direction": "ASC",
                "expression": expression.expression
            });
            return this.order(data, order);
        } else if (descRegex.test(data)) {
            data = expect(data, descRegex);
            data = expect(data, parenthesisLeft);
            expression = this.expression(data);
            data = expect(expression.remainder, parenthesisRight);
            data = ws(data);
            order.push({
                "direction": "DESC",
                "expression": expression.expression
            });
            return this.order(data, order);
        } else if (variableRegex.test(data)) {
            expression = this.expression(data);
            data = ws(expression.remainder);
            order.push({
                "direction": "ASC",
                "expression": expression.expression
            });
            return this.order(data, order);
        }
        return {
            "order": order,
            "remainder": data
        };
    },
    "parse": function (data) {
        return {
            "syntaxTree": SparqlParser.parser.parse(data),
            "remainder": ""
        };
    },
    "prefix": function (data) {
        var prefix = stringRegex.exec(data)[0],
```

```
            local;
        data = data.substr(prefix.length);
        data = expect(data, colonRegex);
        data = ws(data, { required: true });
        data = expect(data, lesserRegex);
        local = uriRegex.exec(data)[0];
        data = data.substr(local.length);
        data = expect(data, greaterRegex);
        return {
            "prefix": token.prefix(prefix, local),
            "remainder": data
        };
    },
    "projection": function (data, projections) {
        projections = projections || [];
        var parsed,
            remainder;
        if (variableRegex.test(data)) {
            parsed = this.variable(data);
            projections.push(parsed.variable);
        } else if (aliasRegex.test(data)) {
            parsed = this.variable(data);
            projections.push(parsed.variable);
        } else if (asteriskRegex.test(data)) {
            parsed = this.variable(data);
            projections.push(parsed.variable);
        } else {
            return {
                "remainder": data,
                "projection": projections
            };
        }
        remainder = ws(parsed.remainder);
        return this.projection(remainder, projections);
    },
    "prologue": function (data, base, prefixes) {
        var prefix;
        base = base || "";
        prefixes = prefixes || [];
        if (baseRegex.test(data)) {
            data = ws(data.substr(4), { required: true });
            data = expect(data, lesserRegex);
            base = this.base(data);
            data = expect(base.remainder, greaterRegex);
            data = ws(data);
            return this.prologue(data, base.base, prefixes);
        } else if (prefixRegex.test(data)) {
            prefix = this.prefix(ws(data.substr(6), { required: true }));
            data = ws(prefix.remainder);
            prefixes.push(prefix.prefix);
            return this.prologue(data, base, prefixes);
        }
        return {
            "prologue": {
                "base": base,
                "prefixes": prefixes,
                "token": "prologue"
            },
            "remainder": data
        };
    },
    /**
     *
     * @param data
     * @param [options]
     * @return {Object}
     */
```

```
        "uri": function (data, options) {
            //console.log(options);
            var uri = {
                "prefix": null,
                "suffix": null,
                "token": "uri",
                "value": null
            },
                remainder,
                value;
            if (lesserRegex.test(data)) {
                data = expect(data, lesserRegex);
                if (uriRegex.test(data)) {
                    uri.value = value = uriRegex.exec(data)[0];
                } else if (options && options.base) {
                    value = stringRegex.exec(data)[0];
                    uri.value = options.base + value;
                } else {
                    throw new Error("IN SPARQL TOKENIZER, URI couldn't parse data: " + data);
                }
                remainder = expect(data.substr(value.length), greaterRegex);
            } else {
                uri.prefix = stringRegex.exec(data)[0];
                remainder = data.substr(uri.prefix.length);
                remainder = expect(remainder, colonRegex);
                uri.suffix = stringRegex.exec(remainder)[0];
                remainder = remainder.substr(uri.suffix.length);
            }
            return {
                "remainder": remainder,
                "uri": uri
            };
        },
        "var": function (data) {
            var tmp = stringRegex.exec(data)[0];
            return {
                "remainder": data.substr(tmp.length),
                "var": token["var"](tmp)
            };
        },
        "variable": function (data) {
            var pVar,
                pVariable,
                expression,
                remainder,
                alias;
            if (variableRegex.test(data)) {
                pVar = this["var"](data.substr(1));
                pVariable = token.variable("var", pVar["var"]);
                return {
                    "remainder": pVar.remainder,
                    "variable": pVariable
                };
            } else if (aliasRegex.test(data)) {
                expression = this.expression(data.substr(1));
                remainder = ws(expression.remainder);
                alias = this.alias(remainder);
                data = expect(alias.remainder, parenthesisRight);
                return {
                    "remainder": data,
                    "variable": {
                        "alias": alias.alias,
                        "expression": expression.expression,
                        "kind": "aliased",
                        "token": "variable"
                    }
                };
```

```
            } else if (asteriskRegex.test(data)) {
                return {
                    "remainder": data.substr(1),
                    "variable": token.variable("*")
                };
            }
            throw new Error("Tried to parse something which is not a variable: " + data);
        },
        /**
         *
         * @param data
         * @param [options]
         * @return {Object}
         */
        where: function (data, options) {
            var bgpindex, pattern, token, value, where;
            data = expect(data, whereRegex);
            data = ws(data);
            data = expect(data, curlyBracketLeftRegex);
            data = ws(data);
            options = Utils.clone(options);
            bgpindex = Utils.isNumber(options.bgpindex) ? options.bgpindex : null;
            pattern = options.pattern;
            if (pattern && pattern.token) {
                token = this.groupgraphpattern(data, pattern.filters, pattern.patterns, bgpindex,
options);
                bgpindex = token.bgpindex;
                data = token.remainder;
                token = token.groupgraphpattern;
            } else if (pattern && pattern.kind) {
                if (variableRegex.test(data) || lesserRegex.test(data)) {
                    token = this.basicgraphpattern(data, [], options);
                    data = token.remainder;
                    value = token.basicgraphpattern.triplesContext;
                    token = whereBGP(value, pattern, options);
                } else if (optionalRegex.test(data)) {
                    token = this.optional(data, options);
                    data = token.remainder;
                    if (pattern.kind === "FILTER") {
                        pattern.value = whereFilter(token, pattern.value);
                        token = pattern;
                    } else {
                        token = whereFilter(token, pattern);
                    }
                } else if (filterRegex.test(data)) {
                    value = this.filter(data, options);
                    data = value.remainder;
                    if (pattern.kind === "FILTER") {
                        token = pattern;
                        token.filter.push(value.filter);
                    } else {
                        token = {
                            "filter": [ value.filter ],
                            "kind": "FILTER",
                            "value": pattern
                        };
                    }
                } else {
                    //console.log("SOMETHING WENT WRONG...");
                    throw new Error("NOT SUPPORTED YET");
                }
                if (!curlyBracketRightRegex.test(data)) {
                    //console.log("SOMETHING WENT WRONG...");
                    throw new Error("NOT SUPPORTED YET");
                }
            } else if (pattern === null) {
                token = this.groupgraphpattern(data, [], [], bgpindex, options);
```

```
                bgpindex = token.bgpindex;
                data = token.remainder;
                token = token.groupgraphpattern;
                //console.log("IN SPARQL, WHERE", bgpindex);
            } else {
                throw new Error("NOT SUPPORTED!");
            }
            data = expect(data, curlyBracketRightRegex);
            data = ws(data);
            //console.log("IN SPARQL, WHERE", token, data);
            where = {
                "remainder": data,
                "where": token
            };
            if (Utils.isNumber(bgpindex)) {
                where.bgpindex = bgpindex;
            }
            return where;
        }
    };
});/*global define*/
define([
    "../loader",
    "../dictionary",
    "../utils",
    "../when"
], function (Loader, Dictionary, Utils, When) {
    "use strict";
    function assignType(predicates, objects, obj) {
        if (Utils.isArray(obj)) {
            Utils.each(obj, function (type) {
                assignType(predicates, objects, type);
            });
        } else {
            predicates.push("rdf:type");
            objects.push(obj);
        }
    }
    var curieRegex = /^[a-zA-Z0-9]+:[a-zA-Z0-9]+/,
        literalStringRegex = /^[a-zA-Z0-9\s:_#\*\$&\-]*/,
        uriRegex = /^http:\/\/[a-zA-Z0-9#_\-.\/]+/,
        ContextLoader = function () {
            return new ContextLoader.prototype.init();
        },
        /**
         *
         * @param pseudoGraph
         * @param properties
         * @param contexts
         * @param options
         * @constructor
         */
        Node = function (pseudoGraph, properties, contexts, bnodes, options) {
            return new Node.prototype.init(pseudoGraph, properties, contexts, bnodes, options);
        },
        /**
         *
         * @param options
         * @constructor
         */
        PseudoGraph = function (options) {
            return new PseudoGraph.prototype.init(options);
        },
        /*
         * The JSON-LD parser object
         *
         * @param {Object} json The JSON-LD to parse
```

```
         * @param {Object} options A given set of options
         * @param {Function} callback A callback-function to run when the graph is assembled
         */
        JSONLD = function (json, options, callback) {
            if (!json || Utils.isNumber(json)) {
                throw new Error("No valid JSON-object given");
            } else if (Utils.isString(json)) {
                try {
                    json = JSON.parse(json);
                } catch (e) {
                    throw new Error("No valid JSON-string given: " + json);
                }
            }
            //console.log("JSON", json);
            var pg = new PseudoGraph(options);
            pg.createNode(json);
            pg.loadContexts(function () {
                callback(pg.assembleTriples());
            });
        };
    ContextLoader.prototype = {
        /**
         * Initialize the ContextLoader
         */
        init: function () {
            this.promises = [];
            this.contexts = [];
        },
        /**
         * Insert a promise to resolve later on
         */
        add: function (promise) {
            return this.promises.push(promise) - 1;
        },
        /**
         * Load all given contexts
         *
         * @param {Function} callback The function to run when all contexts have been loaded
         */
        load: function (callback) {
            var cl = this;

            When.all(this.promises).then(function (contexts) {
                if (contexts) {
                    Utils.each(contexts, function (context) {
                        cl.contexts.push(context);
                    });
                }
                callback();
            }, function (err) {
                throw new Error("There was an error:" + err);
            });
        }
    };
    ContextLoader.prototype.init.prototype = ContextLoader.prototype;
    Node.prototype = {
        /**
         *
         * @param pseudoGraph
         * @param properties
         * @param contexts
         * @param options
         * @return {*}
         */
        init: function (pseudoGraph, properties, contexts, bnodes, options) {
            var promise,
                index,
```

```
            rest,
            obj,
            node = {},
            objects = [],
            predicates = [];
        this.contexts = Utils.clone(contexts);
        this.context = {
            "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
            "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
            "xsd": "http://www.w3.org/TR/xmlschema-2/#"
        };
        this.options = options;
        this.bnodes = bnodes;
        obj = Utils.extract(properties, "@context");
        if (obj) {
            promise = this.getPromise(obj);
            index = pseudoGraph.contextLoader.add(promise);
            this.contexts.push(index);
        }
        obj = Utils.extract(properties, "@type");
        if (obj) {
            assignType(predicates, objects, obj);
        }
        obj = Utils.extract(properties, "@language");
        if (obj) {
            this.lang = obj;
        }
        obj = Utils.extract(properties, "@value");
        if (obj) {
            this.value = obj;
        }
        obj = Utils.extract(properties, "@list");
        if (obj) {
            rest = "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil";
            Utils.each(obj.reverse(), function (obj) {
                node = pseudoGraph.createNode({
                    "rdf:first": obj,
                    "rdf:rest": rest,
                    "rdf:type": "rdf:List"
                });
                rest = node.subject;
            });
            Utils.extend(this, node);
            return undefined;
        }
        this.subject = Utils.extract(properties, "@id");
        if (Utils.size(properties) > 0) {
            this.subject = this.subject || "_:" + Math.random();
            node = this;
            Utils.each(properties, function (obj, property) {
                node.addProperty(pseudoGraph, predicates, objects, property, obj);
            });
        }
        this.triples = [];
        Utils.each(predicates, function (predicate, i) {
            this.triples.push([this.subject, predicate, objects[i]]);
        }.bind(this));
        return this;
    },
    /**
     * Add properties to given arrays
     *
     * @param {Array} predicates An array of predicates
     * @param {Array} objects An array of objects
     * @param {string} property The property to add
     * @param {Object|Array|String} obj The object to add
     */
```

```
    addProperty: function (pseudoGraph, predicates, objects, property, obj) {
        var node,
            object;
        if (!Utils.isObject(obj)) {
            predicates.push(property);
            objects.push(obj);
        } else if (Utils.isArray(obj)) {
            node = this;
            Utils.each(obj, function (o) {
                node.addProperty(pseudoGraph, predicates, objects, property, o);
            });
        } else {
            predicates.push(property);
            if (obj["@value"]) {
                objects.push(pseudoGraph.createNode(obj));
            } else {
                object = pseudoGraph.createNode(obj, this.contexts);
                objects.push(object.subject);
            }
        }
    },
    /**
     * Add the contained triples to the given graph
     *
     * @param {Array} graph The graph to add to
     */
    addTriples: function (graph) {
        var node = this,
            subject,
            predicate,
            object;
        //console.log("TRIPLES ASSEMBLING", this.triples);
        Utils.each(this.triples, function (triple) {
            //console.log("TRIPLE ASSEMBLING", triple);
            subject = node.getSubject(triple[0]);
            if (subject.type === "bnode") {
                subject = node.getBlankNode(subject.value);
            } else if (subject.type === "uri") {
                subject = Dictionary.Symbol(subject.value);
            } else {
                throw new Error("Unrecognized type of subject" + subject.type);
            }
            predicate = node.getPredicate(triple[1]);
            predicate = Dictionary.Symbol(predicate);
            object = node.getObject(triple[2], predicate);
            if (object.type === "bnode") {
                object = node.getBlankNode(object.value);
            } else if (object.type === "uri") {
                object = Dictionary.Symbol(object.value);
            } else {
                //console.log("IN JSONLD, OBJECT", object);
                object = Dictionary.Literal(object.value, object.lang, object.datatype);
            }
            //console.log("TRIPLE", subject, predicate, object);
            graph.add(subject, predicate, object);
        });
    },
    /**
     * Derive contexts
     *
     * @param {Array} contexts
     */
    deriveContexts: function (contexts) {
        var node = this;
        Utils.each(this.contexts, function (num) {
            Utils.extend(node.context, contexts[num]);
        });
```

```
    },
    getBlankNode: function (value) {
        var subject;
        //console.log("BNODES", this.bnodes);
        if (this.bnodes[value]) {
            subject = this.bnodes[value];
        } else {
            subject = Dictionary.BlankNode(value);
            this.bnodes[value] = subject;
        }
        //console.log("BNODE", value, subject);
        return subject;
    },
    /**
     * In case of dereferencing the loader is handy
     *
     * @returns {Object} The loader
     */
    getLoader: function (options) {
        var loader;
        if (this.options && this.options.loader) {
            loader = this.options.loader;
        }
        if (!loader) {
            loader = new Loader(options);
        }
        return loader;
    },
    /**
     * Get the expanded version of the object
     *
     * @param {Object|String} object The object to be expanded
     * @param {String} predicate Sometimes the predicate affects the object
     * @returns {string} The expanded object
     */
    getObject: function (object, predicate) {
        var value, tmp, pre;
        if (Utils.isInteger(object)) {
            object = {
                value: object,
                type: "literal",
                datatype: "http://www.w3.org/TR/xmlschema-2/#integer"
            };
        } else if (Utils.isDouble(object)) {
            object = {
                value: object,
                type: "literal",
                datatype: "http://www.w3.org/TR/xmlschema-2/#double"
            };
        } else if (Utils.isBoolean(object)) {
            object = {
                value: object,
                type: "literal",
                datatype: "http://www.w3.org/TR/xmlschema-2/#boolean"
            };
        } else if (Utils.isString(object) && object[0] === "_") {
            object = {
                value: object,
                type: "bnode"
            };
        } else if (Utils.isString(object)) {
            value = literalStringRegex.exec(object)[0];
            if (uriRegex.test(value)) {
                value = this.getUri(value);
            } else if (curieRegex.test(value)) {
                tmp = this.getUri(value);
                if (uriRegex.test(tmp)) {
```

```
                value = tmp;
            }
        } else {
            value = object;
        }
        object = {
            value: value,
            type: Utils.isUri(value) ? "uri" : "literal"
        };
    }
    if (this.context) {
        object.lang = object.lang || this.context["@language"];
        if (this.context[predicate]) {
            pre = this.context[predicate];
            if (pre["@type"]) {
                pre["@id"] = pre["@id"] || self.getUri(predicate);
                object.datatype = (pre["@type"] !== "@id") ? pre["@type"] : pre["@id"];
            }
        }
    }
    return object;
},
/**
 * Get the expanded version of the predicate
 *
 * @param {String} predicate The predicate to be expanded
 * @returns {string} The expanded predicate
 */
getPredicate: function (predicate) {
    if (this.context) {
        predicate = Utils.isUri(predicate)
            ? predicate
            : this.getUri(predicate);
    }
    return predicate;
},
/**
 * Construct a promise based on the type of object passed
 *
 * @param {String|Array|Object} obj Can be an array, an object or a string
 * @returns {Object} A promise that can be resolved with the Promise Pattern
 */
getPromise: function (obj) {
    var deferred = When.defer();
    if (Utils.isString(obj)) {
        this.getLoader({
            uri: obj,
            success: function (err, result) {
                deferred.resolve(err ? err : JSON.parse(result)["@context"]);
            }
        });
    } else if (Utils.isArray(obj)) {
        var promises = [],
            node = this;
        Utils.each(obj, function (nObj) {
            promises.push(node.getPromise(nObj));
        });
        When.all(promises).then(function () {
            if (arguments[0]) {
                var result = {};
                Utils.each(arguments[0], function (context) {
                    Utils.extend(result, context);
                });
                deferred.resolve(result);
            }
        });
    } else {
```

```
                deferred.resolve(obj);
            }
            return deferred;
        },
        /**
         * Get the expanded subject
         *
         * @param {Object} subject The subject to be expanded
         * @returns {string} The expanded subject
         */
        getSubject: function (subject) {
            subject = this.context && this.context["@id"]
                ? this.context["@id"]
                : subject;
            if (Utils.isString(subject) && subject[0] === "_") {
                subject = {
                    value: subject,
                    type: "bnode"
                };
            } else if (Utils.isString(subject)) {
                subject = {
                    value: this.getUri(subject),
                    type: "uri"
                }
            }
            return subject;
        },
        /**
         * Expand the uri with the help of the context
         * @param {Object} curie The object to be expanded
         * @return {*} The expanded object
         */
        getUri: function (curie) {
            var context = Utils.clone(this.context);
            Utils.each(context, function (obj, key) {
                if (obj.hasOwnProperty("@id")) {
                    context[key] = obj["@id"];
                }
            });
            return Utils.getUri(curie, context);
        }
    };
    Node.prototype.init.prototype = Node.prototype;
    PseudoGraph.prototype = {
        /**
         * Initialize the pseudograph
         *
         * @param {Object} options Given options
         */
        init: function (options) {
            this.contextLoader = ContextLoader();
            this.nodes = [];
            this.options = options;
        },

        /**
         * Assemble a complete graph
         *
         * @returns {Object} The graph
         */
        assembleTriples: function () {
            var graph = Dictionary.Formula(this.options.graph),
                cl = this.contextLoader;
            Utils.each(this.nodes, function (node) {
                //console.log("NODE ASSEMBLING", node);
                node.deriveContexts(cl.contexts);
                node.addTriples(graph);
```

```
            });
            return graph;
        },
        /**
         * Load contexts that is stored in ContextLoader
         *
         * @param {Object} callback The function to run when all contexts has been loaded
         */
        loadContexts: function (callback) {
            this.contextLoader.load(callback);
        },
        /**
         * Traverse nodes in a JSON-LD graph
         *
         * @param {Object} nodes The JSON-object that is to be parsed
         * @param {Array} [contexts] An array with the contexts inherited from the parent (optional)
         * @returns {Object} The node
         */
        createNode: function (nodes, contexts) {
            contexts = contexts || [];
            var pg = this;
            if (Utils.isArray(nodes)) {
                Utils.each(nodes, function (node) {
                    pg.createNode(node, contexts);
                });
                return undefined;
            }
            var n = Node(this, nodes, contexts, this.options);
            if (!n.hasOwnProperty("deriveContexts")) {
                this.nodes.push(n);
            }
            return n;
        }
    };
    PseudoGraph.prototype.init.prototype = PseudoGraph.prototype;
    return JSONLD;
});
/*global define */
define([
    "../dictionary",
    "../utils"
], function (Dictionary, Utils) {
    "use strict";
    return function (json, options, callback) {
        var graph = Dictionary.Formula(options.graph);
        if (!json) {
            throw new Error("No valid JSON-object given");
        } else if (Utils.isString(json)) {
            try {
                json = JSON.parse(json);
            } catch (e) {
                throw new Error("Couldn't parse given rdfjson: " + e);
            }
        }
        Utils.each(json, function (predicates, subject) {
            subject = subject[0] === "_" ?
                    Dictionary.BlankNode(subject) :
                    Dictionary.Symbol(subject);
            Utils.each(predicates, function (objects, predicate) {
                predicate = Dictionary.Symbol(predicate);
                Utils.each(objects, function (object) {
                    if (Utils.isObject(object)) {
                        if (object.type === "literal") {
                            object = Dictionary.Literal(object.value, object.lang, object.datatype);
                        } else {
                            object = Dictionary.Symbol(object.value);
                        }
```

```
                    } else {
                        object = Dictionary.Literal(object);
                    }
                    graph.add(subject, predicate, object);
                });
            });
        });
        callback(graph);
    };
});/*global define */
define([
    "../utils"
], function (Utils) {
    "use strict";
    var sparql = function (syntaxTree) {
        return sparql.assemble(syntaxTree);
    };
    sparql.assemble = function (node) {
        switch (node.token) {
        case "basicgraphpattern":
            return sparql.assembleBasicGraphPattern(node);
        case "blank":
            return sparql.assembleBlank(node);
        case "executableunit":
            return sparql.assembleExecutableUnit(node);
        case "expression":
            return sparql.assembleExpression(node);
        case "filter":
            return sparql.assembleFilter(node);
        case "groupgraphpattern":
            return sparql.assembleGroupGraphPattern(node);
        case "literal":
            return sparql.assembleLiteral(node);
        case "optionalgraphpattern":
            return sparql.assembleOptionalGraphPattern(node);
        case "path":
            return sparql.assemblePath(node);
        case "query":
            return sparql.assembleQuery(node);
        case "uri":
            return sparql.assembleUri(node);
        case "var":
        case "variable":
            return sparql.assembleVariable(node);
        default:
            return node;
        }
    };
    sparql.assembleAdditiveExpression = function (node) {
        return [
            "(",
            sparql.assemble(node.summand),
            Utils.map(node.summands, function (summand) {
                return " {0} {1}".format(summand.operator, sparql.assemble(summand.expression));
            }).join(""),
            ")"
        ].join("");
    };
    sparql.assembleAggregate = function (node) {
        //console.log("assembleAggregate", node);
        return "{0}({1}{2})".format(
            node.aggregateType.toUpperCase(),
            node.distinct ? "DISTINCT " : "",
            sparql.assemble(node.expression)
        );
    };
    sparql.assembleAlias = function (node) {
```

```
        return node.token === "var"
            ? sparql.assembleVariable(node)
            : node.value;
    };
    sparql.assembleAsk = function (node) {
        return "ASK WHERE { {0} }".format(
            sparql.assemble(node.pattern)
        );
    };
    sparql.assembleAtomic = function (node) {
        return "(" + sparql.assemble(node.value) + ")";
    };
    sparql.assembleBase = function (node) {
        if (!node.base) {
            return null;
        }
        return "BASE <{0}>\n".format(node.base.value);
    };
    sparql.assembleBasicGraphPattern = function (node) {
        return sparql.assembleTripleContext(node.triplesContext);
    };
    sparql.assembleBlank = function (node) {
        if (node.value[0] === "_") {
            return "[]";
        }
        return "_:" + node.value;
    };
    sparql.assembleBuiltinCall = function (node) {
        switch (node.builtincall) {
        case "bnode":
            return sparql.assembleBuiltinCallBNode(node);
        case "datatype":
            return sparql.assembleBuiltinCallDatatype(node);
        case "exists":
            return sparql.assembleBuiltinCallExists(node);
        case "if":
            return sparql.assembleBuiltinCallIf(node);
        case "iri":
            return sparql.assembleBuiltinCallIri(node);
        case "lang":
            return sparql.assembleBuiltinCallLang(node);
        case "notexists":
            return sparql.assembleBuiltinCallNotExists(node);
        case "str":
            return sparql.assembleBuiltinCallStr(node);
        case "uri":
            return sparql.assembleBuiltinCallUri(node);
        default:
            throw new Error("assembleBuiltinCall not supported " + node.builtincall);
        }
    };
    sparql.assembleBuiltinCallBNode = function (node) {
        return "BNODE({0})".format(
            Utils.map(node.args, function (arg) {
                return sparql.assemble(arg);
            }).join("")
        );
    };
    sparql.assembleBuiltinCallDatatype = function (node) {
        return "datatype" + sparql.assemble(node.args[0]);
    };
    sparql.assembleBuiltinCallExists = function (node) {
        return "EXISTS {\n{0}\n}\n".format(
            sparql.assemble(node.args[0])
        );
    };
    sparql.assembleBuiltinCallIf = function (node) {
```

```
        return "IF({0}, {1}, {2})".format(
            sparql.assemble(node.args[0]),
            sparql.assemble(node.args[1]),
            sparql.assemble(node.args[2])
        );
    };
    sparql.assembleBuiltinCallIri = function (node) {
        return "IRI{0}".format(
            sparql.assemble(node.args[0])
        );
    };
    sparql.assembleBuiltinCallLang = function (node) {
        return "lang({0})".format(
            sparql.assemble(node.args[0])
        );
    };
    sparql.assembleBuiltinCallNotExists = function (node) {
        return "NOT EXISTS {\n{0}\n}\n".format(
            sparql.assemble(node.args[0])
        );
    };
    sparql.assembleBuiltinCallStr = function (node) {
        return "str" + sparql.assemble(node.args[0]);
    };
    sparql.assembleBuiltinCallUri = function (node) {
        return "URI{0}".format(
            sparql.assemble(node.args[0])
        );
    };
    sparql.assembleConditionalAnd = function (node) {
        return "({0} && {1})".format(
            sparql.assemble(node.operands[0]),
            sparql.assemble(node.operands[1])
        );
    };
    sparql.assembleConditionalOr = function (node) {
        return "({0} || {1})".format(
            sparql.assemble(node.operands[0]),
            sparql.assemble(node.operands[1])
        );
    };
    sparql.assembleCreate = function (node) {
        return "CREATE GRAPH " + sparql.assemble(node.destinyGraph);
    };
    sparql.assembleDelete = function (node) {
        if (!node["delete"]) {
            return null;
        }
        return "DELETE\n{\n" + sparql.assembleTripleContext(node["delete"]) + "\n}\n";
    };
    sparql.assembleDeleteData = function (node) {
        return "DELETE DATA\n{\n{0}\n}\n".format(sparql.assembleTripleContext(node.quads));
    };
    sparql.assembleExecutableUnit = function (node) {
        switch (node.kind) {
        case "ask":
            return sparql.assembleAsk(node);
        case "create":
            return sparql.assembleCreate(node);
        case "deletedata":
            return sparql.assembleDeleteData(node);
        case "load":
            return sparql.assembleLoad(node);
        case "select":
            return sparql.assembleSelect(node);
        case "insertdata":
            return sparql.assembleInsertData(node);
```

```
        default:
            throw new Error("No support for executable kind " + node.kind);
        }
    };
    sparql.assembleExpression = function (node) {
        switch (node.expressionType) {
        case "additiveexpression":
            return sparql.assembleAdditiveExpression(node);
        case "aggregate":
            return sparql.assembleAggregate(node);
        case "atomic":
            return sparql.assembleAtomic(node);
        case "builtincall":
            return sparql.assembleBuiltinCall(node);
        case "conditionaland":
            return sparql.assembleConditionalAnd(node);
        case "conditionalor":
            return sparql.assembleConditionalOr(node);
        case "irireforfunction":
            return sparql.assembleIriRefOrFunction(node);
        case "multiplicativeexpression":
            return sparql.assembleMultiplicativeExpression(node);
        case "relationalexpression":
            return sparql.assembleRelationalExpression(node);
        case "unaryexpression":
            return sparql.assembleUnaryExpression(node);
        default:
            throw new Error("NO SUPPORT FOR expressionType " + node.expressionType);
        }
    };
    sparql.assembleFilter = function (node) {
        //console.log("FILTER", node);
        return "FILTER {0}".format(
            sparql.assemble(node.value)
        );
    };
    sparql.assembleGroup = function (node) {
        if (!node.group || node.group.length === 0) {
            return null;
        }
        return "GROUP BY " + Utils.map(node.group, function (g) {
            return sparql.assemble(g);
        }).join("");
    };
    sparql.assembleGroupGraphPattern = function (node) {
        return [
            Utils.map(node.patterns, function (pattern) {
                return sparql.assemble(pattern);
            }).join("\n"),
            "\n",
            Utils.map(node.filters, function (filter) {
                return sparql.assemble(filter);
            }).join("\n")
        ].join("");
    };
    sparql.assembleInsert = function (node) {
        if (!node.insert) {
            return null;
        }
        return "INSERT\n{\n" + sparql.assembleTripleContext(node.insert) + "\n}\n";
    };
    sparql.assembleInsertData = function (node) {
        return "INSERT DATA\n{\n{0}\n}\n".format(sparql.assembleTripleContext(node.quads));
    };
    sparql.assembleIriRefOrFunction = function (node) {
        return sparql.assembleUri(node.iriref);
    };
```

```
    sparql.assembleLiteral = function (node) {
        if (node.type) {
            return node.value;
        }
        return '"{0}"{1}'.format(
            node.value,
            node.lang ? "@" + node.lang : ""
        );
    };
    sparql.assembleLoad = function (node) {
        //console.log("assembleLoad", node);
        return "LOAD {0}{1}".format(
            sparql.assemble(node.sourceGraph),
            node.destinyGraph ? "INTO GRAPH " + sparql.assemble(node.destinyGraph) : ""
        );
    };
    sparql.assembleMultiplicativeExpression = function (node) {
        return [
            sparql.assemble(node.factor),
            Utils.map(node.factors, function (factor) {
                return " {0} {1}".format(factor.operator, sparql.assemble(factor.expression));
            }).join("")
        ].join("");
    };
    sparql.assembleOptionalGraphPattern = function (node) {
        return " OPTIONAL { " + sparql.assemble(node.value) + " }";
    };
    sparql.assembleOrder = function (node) {
        if (!node.order || node.order.length === 0) {
            return null;
        }
        return "ORDER BY " + Utils.map(node.order, function (order) {
            return "{0}({1}) ".format(
                order.direction,
                sparql.assemble(order.expression)
            );
        }).join(" ");
    };
    sparql.assemblePath = function (node) {
        switch (node.kind) {
        case "alternative":
            return sparql.assemblePathAlternative(node);
        case "element":
            return sparql.assemblePathElement(node);
        case "inversePath":
            return sparql.assemblePathInversePath(node);
        case "sequence":
            return sparql.assemblePathSequence(node);
        default:
            throw new Error("assemblePath not supported " + node.kind);
        }
    };
    sparql.assemblePathAlternative = function () {
        //console.log("assemblePathAlternative", node);
    };
    sparql.assemblePathElement = function (node) {
        //console.log("assemblePathElement", node);
        return "({0}){1}".format(
            sparql.assemble(node.value),
            Utils.isArray(node.modifier)
                ? Utils.map(Utils.flatten(node.modifier), function (mod) {
                    return sparql.assemble(mod);
                }).join("")
                : sparql.assemble(node.modifier)
        );
    };
    sparql.assemblePathInversePath = function (node) {
```

```
        //console.log("assemblePathInversePath", node);
        return "^({0})".format(sparql.assemble(node.value));
    };
    sparql.assemblePathSequence = function (node) {
        //console.log("assemblePathSequence", node);
        return Utils.map(node.value, function (v) {
            return sparql.assemble(v);
        }).join("/");
    };
    sparql.assembleProjection = function (node) {
        //console.log("PROJECTION", node.projection);
        return "SELECT {0}\n".format(
            Utils.map(node.projection, function (project) {
                return sparql.assemble(project);
            }).join(" ")
        );
    };
    sparql.assemblePrologue = function (node) {
        if (!node || !Utils.isObject(node)) {
            return null;
        }
        return [
            sparql.assembleBase(node),
            Utils.map(node.prefixes, function (prefix) {
                return "PREFIX {0}: <{1}>\n".format(prefix.prefix, prefix.local);
            }).join("")
        ].join("");
    };
    sparql.assembleQuery = function (node) {
        return [
            sparql.assemblePrologue(node.prologue),
            "\n",
            Utils.map(node.units, function (unit) {
                if (unit.token) {
                    return sparql.assemble(unit);
                }
                return [
                    sparql.assembleWith(unit),
                    sparql.assembleDelete(unit),
                    sparql.assembleInsert(unit),
                    sparql.assembleUsing(unit),
                    sparql.assembleWhere(unit)
                ].join("");
            }).join("\n;\n")
        ].join("");
    };
    sparql.assembleRelationalExpression = function (node) {
        return "{0} {1} {2}".format(
            sparql.assemble(node.op1),
            node.operator,
            sparql.assemble(node.op2)
        );
    };
    sparql.assembleSelect = function (node) {
        return [
            sparql.assembleProjection(node),
            sparql.assembleWhere(node),
            sparql.assembleGroup(node),
            sparql.assembleOrder(node)
        ].join("");
    };
    sparql.assembleTripleContext = function (tripleContext) {
        var graphName,
            graphs = {};
        Utils.each(tripleContext, function (triple) {
            graphName = triple.graph ? sparql.assemble(triple.graph) : "default";
            //console.log("GRAPH NAME", graphName);
```

```
            if (!graphs[graphName]) {
                graphs[graphName] = [];
            }
            graphs[graphName].push("{0} {1} {2} .".format(
                sparql.assemble(triple.subject),
                sparql.assemble(triple.predicate),
                sparql.assemble(triple.object)
            ));
        });
        return Utils.map(graphs, function (triples, graph) {
            if (graph !== "default") {
                return "GRAPH {0} {{1}}".format(graph, triples.join(""));
            }
            return triples.join("");
        }).join(" . ");
    };
    sparql.assembleUnaryExpression = function (node) {
        return "(! {0})".format(
            sparql.assemble(node.expression)
        );
    };
    sparql.assembleUri = function (node) {
        if (node.value) {
            return "<{0}>".format(node.value);
        }
        return node.prefix + ":" + node.suffix;
    };
    sparql.assembleUsing = function (node) {
        if (!node.using) {
            return null;
        }
        return Utils.map(node.using, function (n) {
            switch (n.kind) {
            case "named":
                return sparql.assembleUsingNamed(n);
            case "default":
                return sparql.assembleUsingDefault(n);
            default:
                //console.log(n);
                throw new Error("NOT SUPPORTING USING " + n.kind);
            }
        }).join("");
    };
    sparql.assembleUsingDefault = function (node) {
        return "USING " + sparql.assemble(node.uri) + "\n";
    };
    sparql.assembleUsingNamed = function (node) {
        return "USING NAMED " + sparql.assemble(node.uri) + "\n";
    };
    sparql.assembleVariable = function (node) {
        switch (node.kind) {
        case "aliased":
            return "({0} AS {1})".format(sparql.assembleExpression(node.expression), sparql.assembleAlias
(node.alias));
        case "var":
            return sparql.assemble(node.value);
        case "*":
            return node.kind;
        default:
            //console.log("VARIABLE", node);
            return "?" + node.value;
        }
    };
    sparql.assembleWhere = function (node) {
        if (!node.pattern) {
            return null;
        }
```

```
        return [
            "WHERE\n{\n",
            sparql.assemble(node.pattern),
            "\n}"
        ].join("");
    };
    sparql.assembleWith = function (node) {
        if (!node["with"]) {
            return null;
        }
        return "WITH " + sparql.assemble(node["with"]) + " ";
    };
    return sparql;
});define([
    "../../graphite/dictionary",
    "../rdf",
    "../uri",
    "../../graphite/utils"
], function (Dictionary, RDF, URI, Utils) {
    /*
     * jQuery RDF @VERSION
     *
     * Copyright (c) 2008,2009 Jeni Tennison
     * Licensed under the MIT (MIT-LICENSE.txt)
     *
     * Depends:
     *  jquery.uri.js
     *  jquery.xmlns.js
     *  jquery.datatype.js
     *  jquery.curie.js
     *  jquery.rdf.js
     *  jquery.rdf.json.js
     *  jquery.rdf.xml.js
     *
     * @fileOverview jQuery RDF/XML parser
     * @author <a href="mailto:jeni@jenitennison.com">Jeni Tennison</a>
     * @copyright (c) 2008,2009 Jeni Tennison
     * @license MIT license (MIT-LICENSE.txt)
     * @version 1.0
     */
    var rdfNs = "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
        getDefaultNamespacePrefix = function (namespaceUri) {
            switch (namespaceUri) {
                case 'http://www.w3.org/1999/02/22-rdf-syntax-ns':
                    return 'rdf';
                case 'http://www.w3.org/XML/1998/namespace':
                    return 'xml';
                case 'http://www.w3.org/2000/xmlns/':
                    return 'xmlns';
                default:
                    throw ('No default prefix mapped for namespace ' + namespaceUri);
            }
        },
        hasAttributeNS  = function(elem, namespace, name){
            var basename;
            if (elem.hasAttributeNS) {
                return elem.hasAttributeNS(namespace, name);
            } else {
                try {
                    basename = /:/.test(name) ? /:(.+)$/.exec(name)[1] : name;
                    return elem.attributes.getQualifiedItem(basename, namespace) !== null;
                } catch (e) {
                    return elem.getAttribute(getDefaultNamespacePrefix(namespace) + ':' + name) !== null;
                }
            }
        },
        getAttributeNS = function(elem, namespace, name){
```

```
        var basename
        //console.log("TEST", elem.getAttributeNS);
        if (elem.getAttributeNS) {
            return elem.getAttributeNS(namespace, name);
        } else {
            try {
                basename = /:/.test(name) ? /:(.+)$/.exec(name)[1] : name;
                return elem.attributes.getQualifiedItem(basename, namespace).nodeValue;
            } catch (e) {
                var tmp = getDefaultNamespacePrefix(namespace) + ':' + name;

                return elem.getAttribute(getDefaultNamespacePrefix(namespace) + ':' + name);
            }
        }
    },
    getLocalName = function(elem){
        return elem.localName || elem.baseName;
    },
    parseRdfXmlSubject = function (elem, base) {
        var s, subject;
        if (hasAttributeNS(elem, rdfNs, 'about')) {
            s = getAttributeNS(elem, rdfNs, 'about');
            subject = Dictionary.Symbol(RDF.resource('<' + s + '>', { base: base }).value);
        } else if (hasAttributeNS(elem, rdfNs, 'ID')) {
            s = getAttributeNS(elem, rdfNs, 'ID');
            subject = Dictionary.Symbol(RDF.resource('<#' + s + '>', { base: base }).value);
        } else if (hasAttributeNS(elem, rdfNs, 'nodeID')) {
            s = getAttributeNS(elem, rdfNs, 'nodeID');
            subject = Dictionary.BlankNode(s);
        } else {
            subject = Dictionary.BlankNode();
        }
        return subject;
    },
    parseRdfXmlDescription = function (elem, isDescription, graph, base, lang) {
        var subject,
            p,
            property,
            o,
            object,
            reified,
            i,
            j,
            li = 1,
            collection1,
            collection2,
            collectionItem,
            collectionItems = [],
            parseType,
            serializer,
            literalOpts = {},
            oTriples,
            triples = [],
            tmpObject1,
            tmpObject2;
        lang = getAttributeNS(elem, 'http://www.w3.org/XML/1998/namespace', 'lang') || lang;
        base = getAttributeNS(elem, 'http://www.w3.org/XML/1998/namespace', 'base') || base;
        if (lang !== null && lang !== undefined && lang !== '') {
            literalOpts = { lang: lang };
        }
        subject = parseRdfXmlSubject(elem, base);
        if (isDescription && (elem.namespaceURI !== rdfNs || getLocalName(elem) !== 'Description')) {
            property = Dictionary.Symbol(RDF.type.value);
            object = Dictionary.Symbol(elem.namespaceURI + getLocalName(elem));
            graph.add(subject, property, object);
        }
        for (i = 0; i < elem.attributes.length; i += 1) {
```

```
                p = elem.attributes.item(i);
                if (p.namespaceURI !== undefined &&
                    p.namespaceURI !== 'http://www.w3.org/2000/xmlns/' &&
                    p.namespaceURI !== 'http://www.w3.org/XML/1998/namespace' &&
                    p.prefix !== 'xmlns' &&
                    p.prefix !== 'xml') {
                    if (p.namespaceURI !== rdfNs) {
                        property = Dictionary.Symbol(p.namespaceURI + getLocalName(p));
                        object = Dictionary.Literal(literalOpts.lang ? p.nodeValue : '"' +
p.nodeValue.replace(/"/g, '\\"'), literalOpts.lang);
                        graph.add(subject, property, object);
                    } else if (getLocalName(p) === 'type') {
                        property = Dictionary.Symbol(RDF.type.value);
                        object = Dictionary.Symbol(RDF.resource('<' + p.nodeValue + '>', { base: base }
).value);
                        graph.add(subject, property, object);
                    }
                }
            }
            var parentLang = lang;
            for (i = 0; i < elem.childNodes.length; i += 1) {
                p = elem.childNodes[i];
                if (p.nodeType === 1) {
                    if (p.namespaceURI === rdfNs && getLocalName(p) === 'li') {
                        property = Dictionary.Symbol(rdfNs + '_' + li);
                        li += 1;
                    } else {
                        property = Dictionary.Symbol(p.namespaceURI + getLocalName(p));
                    }
                    lang = getAttributeNS(p, 'http://www.w3.org/XML/1998/namespace', 'lang') ||
parentLang;
                    if (lang !== null && lang !== undefined && lang !== '') {
                        literalOpts = { lang: lang };
                    } else {
                        literalOpts = {};
                    }
                    if (hasAttributeNS(p, rdfNs, 'resource')) {
                        o = getAttributeNS(p, rdfNs, 'resource');
                        object = Dictionary.Symbol(RDF.resource('<' + o + '>', { base: base }).value);
                    } else if (hasAttributeNS(p, rdfNs, 'nodeID')) {
                        o = getAttributeNS(p, rdfNs, 'nodeID');
                        object = Dictionary.BlankNode(o);
                    } else if (hasAttributeNS(p, rdfNs, 'parseType')) {
                        parseType = getAttributeNS(p, rdfNs, 'parseType');
                        if (parseType === 'Literal') {
                            try {
                                serializer = new XMLSerializer();
                                o = serializer.serializeToString(p.getElementsByTagName('*')[0]);
                            } catch (e) {
                                o = "";
                                for (j = 0; j < p.childNodes.length; j += 1) {
                                    o += p.childNodes[j].xml;
                                }
                            }
                            object = Dictionary.Literal(o, null, Dictionary.Symbol(rdfNs + 'XMLLiteral'));
                        } else if (parseType === 'Resource') {
                            tmpObject1 = Utils.last(graph.statements);
                            parseRdfXmlDescription(p, false, graph, base, lang);
                            tmpObject2 = Utils.last(graph.statements);
                            //console.log("TMP OBJECT 1", tmpObject1);
                            if (tmpObject1 && tmpObject1 !== tmpObject2) {
                                object = tmpObject2;
                            } else {
                                object = Dictionary.BlankNode();
                            }
                        } else if (parseType === 'Collection') {
                            if (p.getElementsByTagName('*').length > 0) {
```

```
                                for (j = 0; j < p.childNodes.length; j += 1) {
                                    o = p.childNodes[j];
                                    if (o.nodeType === 1) {
                                        collectionItems.push(o);
                                    }
                                }
                                collection1 = Dictionary.BlankNode();
                                object = collection1;
                                for (j = 0; j < collectionItems.length; j += 1) {
                                    o = collectionItems[j];
                                    tmpObject1 = Utils.last(graph.statements);
                                    parseRdfXmlDescription(o, true, graph, base, lang);
                                    tmpObject2 = Utils.last(graph.statements);
                                    //console.log("TMP OBJECT 2", tmpObject1);
                                    if (tmpObject1 && tmpObject1 !== tmpObject2) {
                                        collectionItem = tmpObject2;
                                    } else {
                                        collectionItem = parseRdfXmlSubject(o);
                                    }
                                    graph.add(collection1, Dictionary.Symbol(RDF.first.value),
collectionItem);
                                    if (j === collectionItems.length - 1) {
                                        graph.add(collection1, Dictionary.Symbol(RDF.rest.value),
Dictionary.Symbol(RDF.nil.value));
                                    } else {
                                        collection2 = Dictionary.BlankNode();
                                        graph.add(collection1, Dictionary.Symbol(RDF.rest.value),
collection2);
                                        collection1 = collection2;
                                    }
                                }
                            } else {
                                object = RDF.nil;
                            }
                        }
                    } else if (hasAttributeNS(p, rdfNs, 'datatype')) {
                        o = p.childNodes[0] ? p.childNodes[0].nodeValue : "";
                        object = Dictionary.Literal(o, null, Dictionary.Symbol(getAttributeNS(p, rdfNs,
'datatype')));
                    } else if (p.getElementsByTagName('*').length > 0) {
                        for (j = 0; j < p.childNodes.length; j += 1) {
                            o = p.childNodes[j];
                            if (o.nodeType === 1) {
                                tmpObject1 = Utils.last(graph.statements);
                                parseRdfXmlDescription(o, true, graph, base, lang);
                                tmpObject2 = Utils.last(graph.statements);
                                //console.log("TMP OBJECT 3", tmpObject1);
                                if (tmpObject1 && tmpObject1 !== tmpObject2) {
                                    object = tmpObject2;
                                } else {
                                    object = parseRdfXmlSubject(o);
                                }
                            }
                        }
                    } else if (p.childNodes.length > 0) {
                        o = p.childNodes[0].nodeValue;
                        object = Dictionary.Literal(literalOpts.lang ? o : '"' + o.replace(/"/g, '\\"') +
'"', literalOpts.lang);
                    } else {
                        tmpObject1 = Utils.last(graph.statements);
                        parseRdfXmlDescription(p, false, graph, base, lang);
                        tmpObject2 = Utils.last(graph.statements);
                        //console.log("TMP OBJECT 4", tmpObject1, graph.statements, tmpObject2);
                        if (tmpObject1 && tmpObject1 !== tmpObject2) {
                            object = tmpObject2;
                        } else {
                            object = Dictionary.BlankNode();
```

```
                        }
                    }
                    graph.add(subject, property, object);
                    if (hasAttributeNS(p, rdfNs, 'ID')) {
                        reified = Dictionary.Symbol(RDF.resource('<#' + getAttributeNS(p, rdfNs, 'ID') +
'>', { base: base }).value);
                        graph.add(reified, Dictionary.Symbol(RDF.subject.value), subject);
                        graph.add(reified, Dictionary.Symbol(RDF.property.value), property);
                        graph.add(reified, Dictionary.Symbol(RDF.object.value), object);
                    }
                }
            }
            return graph;
        },
        parseRdfXml = function (doc, options) {
            options = options || {};
            var base,
                lang,
                graph = Dictionary.Formula(options.graph);
            if (doc.documentElement.namespaceURI === rdfNs && getLocalName(doc.documentElement) ===
'RDF') {
                lang = getAttributeNS(doc.documentElement, 'http://www.w3.org/XML/1998/namespace',
'lang');
                base = getAttributeNS(doc.documentElement, 'http://www.w3.org/XML/1998/namespace',
'base') || URI.base();
                Utils.each(doc.documentElement.childNodes, function (d) {
                    if (d.nodeType === 1) {
                        parseRdfXmlDescription(d, true, graph, base, lang);
                    }
                });
            } else {
                parseRdfXmlDescription(doc.documentElement, true, graph);
            }
            return graph;
        };
    return function (data, options, callback) {
        var doc;
        try {
            doc = new ActiveXObject("Microsoft.XMLDOM");
            doc.async = "false";
            doc.loadXML(data);
        } catch(e) {
            var parser = new DOMParser();
            doc = parser.parseFromString(data, 'text/xml');
        }
        callback(parseRdfXml(doc, options));
    };
});define([
    "../../graphite/dictionary",
    "../rdf",
    "../uri"
], function(Dictionary, RDF, URI) {
    var wsRegex = /^(\u0009|\u000A|\u000D|\u0020|#([^\u000A\u000D])*)+/,
        nameStartChars = 'A-Z_a-z\u00C0-\u00D6\u00D8-\u00F6\u00F8-\u02FF\u0370-\u037D\u037F-\u1FFF\u200C-
\u200D\u2070-\u218F\u2C00-\u2FEF\u3001-\uD7FF\uF900-\uFDCF\uFDF0-\uFFFD', // can't include \u10000-\uEFFFF
        nameChars = '-' + nameStartChars + '0-9\u00B7\u0300-\u036F\u203F-\u2040',
        nameRegex = new RegExp('^[' + nameStartChars + '][' + nameChars + ']*'),
        uriRegex = /^(\\u[0-9A-F]{4}|\\U[0-9A-F]{8}|\\\\|\\>|[\u0020-\u003D\u003F-\u005B\u005D-
\u10FFFF])*/,
        booleanRegex = /^(true|false)[\s\.;,)\]]/,
        doubleRegex = /^(-|\+)?([0-9]+\.[0-9]*[eE](-|\+)?[0-9]+|\.[0-9]+[eE](-|\+)?[0-9]+|[0-9]+[eE](-|\
+)?[0-9]+)/,
        decimalRegex = /^(-|\+)?(([0-9]+\.[0-9]*)|(\.[0-9]))+/,
        integerRegex = /^(-|\+)?[0-9]+/,
        stringRegex = /^(\\u[0-9A-F]{4}|\\U[0-9A-F]{8}|\\\\|[\u0020-\u0021\u0023-\u005B]|[\u005D-
\u10FFFF]|\\t|\\n|\\r|\\")*/,
        longStringRegex = /^(\\u[0-9A-F]{4}|\\U[0-9A-F]{8}|\\\\|[\u0020-\u0021\u0023-\u005B]|[\u005D-
```

```
\u10FFFF]|\\t|\\n|\\r|\\"|\u0009|\u000A|\u000D|"[^"]|""[^"])*/,
        languageRegex = /^[a-z]+(-[a-z0-9]+)*/;
    function blank (data, graph, opts) {
        var parsed,
            bnode,
            first = data.substring(0, 1);
        //log('blank: ' + data);
        if (first === '_') {
            data = validate(data, '_:');
            parsed = name(data);
            return {
                remainder: parsed.remainder,
                blank: Dictionary.BlankNode(parsed.name)
            }
        } else if (first === '(') {
            parsed = collection(data, graph, opts);
            return {
                remainder: parsed.remainder,
                blank: parsed.collection
            };
        } else if (data.substring(0, 2) === '[]') {
            return {
                remainder: data.substring(2),
                blank: Dictionary.BlankNode()
            };
        } else {
            bnode = Dictionary.BlankNode();
            opts.subject.unshift(bnode);
            data = validate(data, '[');
            data = ws(data);
            parsed = predicateObjectList(data, graph, opts);
            data = parsed.remainder;
            data = ws(data);
            data = validate(data, ']');
            opts.subject.shift();
            return {
                remainder: data,
                blank: bnode
            };
        }
    }
    function collection (data, graph, opts) {
        var parsed,
            i,
            items,
            list,
            rest = Dictionary.Symbol(RDF.nil.value);
        //log('collection: ' + data);
        data = validate(data, '(');
        data = ws(data);
        parsed = itemList(data, graph, opts);
        data = parsed.remainder;
        items = parsed.items;
        for (i = items.length - 1; i >= 0; i -= 1) {
            list = Dictionary.BlankNode();
            graph.add(list, Dictionary.Symbol(RDF.first.value), items[i]);
            graph.add(list, Dictionary.Symbol(RDF.rest.value), rest);
            rest = list;
        }
        data = ws(data);
        data = validate(data, ')');
        return {
            remainder: data,
            collection: rest
        };
    }
    function directive (data, opts) {
```

```
        var parsed, prefix;
        //log('directive: ' + data);
        if (data.substring(0, 7) === '@prefix') {
            data = data.substring(7);
            data = ws(data, { required: true });
            parsed = prefixName(data);
            prefix = parsed.prefix;
            data = parsed.remainder;
            data = ws(data);
            data = validate(data, ':');
            data = ws(data);
            parsed = uriref(data, opts);
            opts.namespaces[prefix] = parsed.uri;
            data = parsed.remainder;
        } else if (data.substring(0, 5) === '@base') {
            data = validate(data, '@base');
            data = ws(data, { required: true });
            parsed = uriref(data, opts);
            opts.base = parsed.uri;
            data = parsed.remainder;
        } else {
            throw ("Invalid Turtle: Unrecognised directive: " + data);
        }
        data = ws(data);
        data = validate(data, '.');
        return {
            remainder: data,
            opts: opts
        };
    }
    function itemList (data, graph, opts) {
        var parsed,
            items = [],
            first = data.substring(0, 1);
        //log('itemList: ' + data);
        while (first !== ')') {
            parsed = object(data, graph, opts);
            data = parsed.remainder;
            items.push(parsed.object);
            data = ws(data);
            first = data.substring(0, 1);
        }
        return {
            remainder: data,
            items: items
        };
    }
    function language (data) {
        var lang;
        //log('language: ' + data);
        lang = languageRegex.exec(data)[0];
        return {
            remainder: data.substring(lang.length),
            language: lang
        };
    }
    function literal (data, opts) {
        var first, str, parsed;
        //log('literal: ' + data);
        if (booleanRegex.test(data)) {
            str = booleanRegex.exec(data)[1];
            return {
                remainder: data.substring(str.length),
                literal: Dictionary.Literal(str, null, Dictionary.Symbol.XSDboolean)
            }
        } else if (doubleRegex.test(data)) {
            str = doubleRegex.exec(data)[0];
```

```
            return {
                remainder: data.substring(str.length),
                literal: Dictionary.Literal(str, null, Dictionary.Symbol.XSDfloat)
            };
        } else if (decimalRegex.test(data)) {
            str = decimalRegex.exec(data)[0];
            return {
                remainder: data.substring(str.length),
                literal: Dictionary.Literal(str, null, Dictionary.Symbol.XSDdecimal)
            };
        } else if (integerRegex.test(data)) {
            str = integerRegex.exec(data)[0];
            return {
                remainder: data.substring(str.length),
                literal: Dictionary.Literal(str, null, Dictionary.Symbol.XSDinteger)
            };
        } else {
            parsed = quotedString(data);
            data = parsed.remainder;
            str = parsed.string;
            data = ws(data);
            first = data.substring(0, 1);
            if (first === '^') {
                data = validate(data, '^^');
                data = ws(data);
                parsed = resource(data, opts);
                return {
                    remainder: parsed.remainder,
                    literal: Dictionary.Literal(str, null, parsed.resource)
                };
            } else if (first === '@') {
                data = validate(data, '@');
                data = ws(data);
                parsed = language(data);
                return {
                    remainder: parsed.remainder,
                    literal: Dictionary.Literal(str, parsed.language)
                };
            } else {
                return {
                    remainder: data,
                    literal: Dictionary.Literal(str)
                };
            }
        }
    }
    function name (data) {
        var result;
        //log('name: ' + data);
        if (nameRegex.test(data)) {
            result = nameRegex.exec(data);
            return {
                name: result[0],
                remainder: data.substring(result[0].length)
            };
        } else {
            return {
                name: '',
                remainder: data
            }
        }
    }
    function object (data, graph, opts) {
        var parsed,
            first = data.substring(0, 1);
        //log('object: ' + data);
        if (first === '[' || first === '_' || first === '(') {
```

```
            parsed = blank(data, graph, opts);
            return {
                remainder: parsed.remainder,
                object: parsed.blank
            };
        } else {
            try {
                parsed = literal(data, opts);
                return {
                    remainder: parsed.remainder,
                    object: parsed.literal
                };
            } catch (e) {
                parsed = resource(data, opts);
                return {
                    remainder: parsed.remainder,
                    object: parsed.resource
                };
            }
        }
    }
    function objectList (data, graph, opts) {
        var parsed,
            first = data.substring(0, 1);
        //log('objectList: ' + data);
        do {
            parsed = object(data, graph, opts);
            data = parsed.remainder;
            graph.add(opts.subject[0], opts.verb[0], parsed.object);
            data = ws(data);
            first = data.substring(0, 1);
            if (first === ',') {
                data = validate(data, ',');
                data = ws(data);
                first = data.substring(0, 1);
            } else {
                break;
            }
        } while (first !== ']' && first !== ';' && first !== '.');
        return {
            remainder: data
        };
    }
    function parse(data, opts) {
        var parsed = {},
            graph = Dictionary.Formula(opts.graph);
        opts = opts || {};
        opts.namespaces = {};
        opts.base = opts.base || URI.base();
        opts.subject = [];
        opts.verb = [];
        while(data !== '') {
            //log('parseTurtle: ' + data);
            parsed = statement(data, graph, opts);
            data = parsed.remainder;
            opts = parsed.opts;
            //triples = triples.concat(parsed.triples);
        }
        return graph;
    }
    function prefixName (data) {
        var n = name(data);
        //log('prefixName: ' + data);
        if (n.name.substring(0, 1) === '_') {
            throw "Invalid Turtle: Prefix must not start with an underscore: " + name;
        } else {
            return {
```

```
            prefix: n.name,
            remainder: n.remainder
        };
    }
}
function predicateObjectList (data, graph, opts) {
    var parsed,
        property,
        first = data.substring(0, 1);
    //log('predicateObjectList: ' + data);
    do {
        parsed = verb(data, opts);
        data = parsed.remainder;
        property = parsed.verb;
        data = ws(data);
        opts.verb.unshift(property);
        parsed = objectList(data, graph, opts);
        data = parsed.remainder;
        opts.verb.shift();
        data = ws(data);
        first = data.substring(0, 1);
        if (first === ';') {
            data = validate(data, ';');
            data = ws(data);
            first = data.substring(0, 1);
        } else {
            break;
        }
    } while (first !== ']' && first !== '.');
    return {
        remainder: data
    };
}
function quotedString (data) {
    var str;
    //log('quotedString: ' + data);
    if (data.substring(0, 3) === '"""') {
        data = validate(data, '"""');
        str = longStringRegex.exec(data)[0];
        data = data.substring(str.length);
        str = str
            .replace(/\n/g, '\\n')
            .replace(/\t/g, '\\t')
            .replace(/\r/g, '\\r')
            .replace(/\\"/g, '"');
        str = unescape(str);
        data = validate(data, '"""');
        return {
            remainder: data,
            string: str
        };
    } else {
        data = validate(data, '"');
        str = stringRegex.exec(data)[0];
        data = data.substring(str.length);
        str = str.replace(/\\"/g, '"');
        str = unescape(str);
        data = validate(data, '"');
        return {
            remainder: data,
            string: str
        };
    }
}
function validate (data, str) {
    if (data.substring(0, str.length) === str) {
        return data.substring(str.length);
```

```
        } else {
            throw "Invalid Turtle: Expecting '" + str + "', found '" + data.substring(0, 20) + "...'";
        }
    }
    function resource (data, opts) {
        var parsed,
            prefix,
            local,
            first = data.substring(0, 1),
            resource;
        //log('resource: ' + data);
        if (first === '<') {
            parsed = uriref(data, opts);
            resource = RDF.resource(parsed.uri, opts.base);
            return {
                remainder: parsed.remainder,
                resource: Dictionary.Symbol(resource.value)
            };
        } else {
            try {
                parsed = prefixName(data);
                prefix = parsed.prefix;
                data = parsed.remainder;
            } catch (e) {
                prefix = '';
            }
            data = validate(data, ':');
            parsed = name(data);
            local = parsed.name;
            resource = RDF.resource(prefix + ':' + local, { namespaces: opts.namespaces, base: opts.base }
);
            return {
                remainder: parsed.remainder,
                resource: Dictionary.Symbol(resource.value)
            };
        }
    }
    function statement(data, graph, opts) {
        var first;
        //log('statement: ' + data);
        data = ws(data);
        if (data.length === 0) {
            return { remainder: '', opts: opts };
        } else {
            first = data.substring(0, 1);
            if (first === '@') {
                return directive(data, opts);
            } else {
                return triples(data, graph, opts);
            }
        }
    }
    function subject(data, graph, opts) {
        var parsed,
            first = data.substring(0, 1);
        //log('subject: ' + data);
        if (first === '[' || first === '_' || first === '(') {
            parsed = blank(data, graph, opts);
            return {
                remainder: parsed.remainder,
                subject: parsed.blank
            };
        } else {
            parsed = resource(data, opts);
            return {
                remainder: parsed.remainder,
                subject: parsed.resource
```

```
            };
        }
    }
    function triples(data, graph, opts) {
        var parsed;
        //log('triples: ' + data);
        parsed = subject(data, graph, opts);
        data = parsed.remainder;
        opts.subject.unshift(parsed.subject);
        data = ws(data);
        parsed = predicateObjectList(data, graph, opts);
        opts.subject.shift();
        data = ws(parsed.remainder);
        data = validate(data, '.');
        return {
            remainder: data,
            opts: opts
        };
    }
    function unescape (string) {
        return string.replace(/(\\u([0-9A-F]{4}))|(\\U([0-9A-F]{4})([0-9A-F]{4}))/g, function (m, u4,
u4h, u8, u8h1, u8h2) {
            if (u4 !== undefined) {
                return String.fromCharCode(parseInt(u4h, 16));
            } else {
                return String.fromCharCode(parseInt(u8h1, 16)) + String.fromCharCode(parseInt(u8h2, 16));
            }
        });
    }
    function uriref (data, opts) {
        var uri;
        //log('uriref: ' + data);
        data = validate(data, '<');
        uri = uriRegex.exec(data)[0];
        data = data.substring(uri.length);
        data = validate(data, '>');
        return {
            remainder: data,
            uri: URI.resolve(uri, opts.base)
        };
    }
    function verb (data, opts) {
        var parsed,
            first = data.substring(0, 1);
        //log('verb: ' + data);
        try {
            parsed = resource(data, opts);
            return {
                remainder: parsed.remainder,
                verb: parsed.resource
            };
        } catch (e) {
            if (first === 'a') {
                data = ws(data.substring(1), { required: true });
                return {
                    remainder: data,
                    verb: Dictionary.Symbol(RDF.type.value)
                };
            } else {
                throw e;
            }
        }
    }
    function ws(data, opts) {
        var required;
        opts = opts || {};
        required = opts.required || false;
```

```
        if (required && !wsRegex.test(data)) {
            throw("Invalid Turtle: Required whitespace is missing!");
        }
        return data.replace(wsRegex, '');
    }
    return function(data, options, callback) {
        if (!data) {
            throw "No valid data was given";
        }
        callback(parse(data, options));
    };
});define([
    "../../graphite/loader",
    "../../graphite/rdfparser",
    "../trees/utils",
    "../../graphite/utils"
], function (Loader, Parser, TreeUtils, Utils) {
    function getMime(responseHeader) {
        return responseHeader.split("application/")[1];
    }

    var RDFLoader = {};
    RDFLoader.RDFLoader = function () {
    //RDFLoader.RDFLoader = function (params) {
        /*
        var that = this;
        this.precedences = ["text/turtle", "text/n3", "application/json"];
        this.parsers = {
            "text/turtle": N3Parser.parser,
            "text/n3": N3Parser.parser,
            "application/json": JSONLDParser.parser
        };
        if (params != null) {
            TreeUtils.each(params["parsers"], function (mime) {
                that.parsers[mime] = params["parsers"][mime];
            });
        }
        if (params && params["precedences"] != null) {
            this.precedences = params["precedences"];
            TreeUtils.each(params["parsers"], function (mime) {
                if (!Utils.include(that.precedences, mime)) {
                    that.precedences.push(mime);
                }
            });
        }
        this.acceptHeaderValue = "";
        for (var i = 0; i < this.precedences.length; i++) {
            if (i != 0) {
                this.acceptHeaderValue = this.acceptHeaderValue + "," + this.precedences[i];
            } else {
                this.acceptHeaderValue = this.acceptHeaderValue + this.precedences[i];
            }
        }
        */
    };
    /*
    RDFLoader.RDFLoader.prototype.registerParser = function(mediaType, parser) {
        this.parsers[mediaType] = parser;
        this.precedences.push(mediaType);
    };
    RDFLoader.RDFLoader.prototype.unregisterParser = function(mediaType) {
        delete this.parsers[mediaType];
        var mediaTypes = [];
        for(var i=0; i<this.precedences.length; i++) {
            if(this.precedences[i] != mediaType) {
                mediaTypes.push(this.precedences[i]);
            }
```

```
        }

        this.precedences = mediaTypes;
    };
    RDFLoader.RDFLoader.prototype.setAcceptHeaderPrecedence = function(mediaTypes) {
        this.precedences = mediaTypes;
    };
     */
    RDFLoader.RDFLoader.prototype.load = function(uri, graph, callback) {
        //console.debug(graph);
        Loader({
            uri: uri,
            accept: this.acceptHeaderValue,
            success: function (err, data, status, xhr) {
                //console.log("RDFLOADER LOAD", data);
                if(!err) {
                    //console.log("DATA RETRIEVED");
                    var mime = getMime(xhr.getResponseHeader("Content-Type") || xhr.getResponseHeader
("content-type"));
                    if(!mime || mime === "octet-stream") {
                        mime = Utils.last(uri.split("."));
                    }
                    //console.log("MIME", mime);
                    Parser(data, mime, {
                        graph: graph
                    }, function (graph) {
                        //console.log("GRAPH", graph, graph.toQuads);
                        callback(true, graph);
                    });
                } else {
                    callback(false, "Network error");
                }
            }
        });
    };
    RDFLoader.RDFLoader.prototype.tryToParse = function(parser, graph, input, callback) {
        try {
            if(typeof(input) === 'string') {
                input = TreeUtils.normalizeUnicodeLiterals(input);
            }
            var parsed = parser.parse(input, graph);

            if(parsed != null) {
                callback(true, parsed);
            } else {
                callback(false, "parsing error");
            }
        } catch(e) {
            //console.log(e.message);
            //console.log(e.stack);
            callback(false, "parsing error with mime type : " + e);
        }
    };
    return RDFLoader;
});define([
    "./../../graphite/queryparser",
    "./../trees/utils",
    "./../../graphite/utils"
], function (QueryParser, TreeUtils, Utils) {
    var AbstractQueryTree = {};
    /**
     * @doc
     *
     * Based on <http://www.w3.org/2001/sw/DataAccess/rq23/rq24-algebra.html>
     * W3C's note
     */
    AbstractQueryTree.AbstractQueryTree = function() {
```

```
};
AbstractQueryTree.AbstractQueryTree.prototype.parseQueryString = function(query_string) {
    //noinspection UnnecessaryLocalVariableJS,UnnecessaryLocalVariableJS
    return QueryParser("sparql").parse(query_string).syntaxTree;
};
AbstractQueryTree.AbstractQueryTree.prototype.parseExecutableUnit = function(executableUnit) {
    if(executableUnit.kind === 'select') {
        return this.parseSelect(executableUnit);
    } else if(executableUnit.kind === 'ask') {
        return this.parseSelect(executableUnit);
    } else if(executableUnit.kind === 'modify') {
        return this.parseSelect(executableUnit);
    } else if(executableUnit.kind === 'construct') {
        return this.parseSelect(executableUnit);
    } else if(executableUnit.kind === 'insertdata') {
        return this.parseInsertData(executableUnit);
    } else if(executableUnit.kind === 'deletedata') {
        return this.parseInsertData(executableUnit);
    } else if(executableUnit.kind === 'load') {
        return executableUnit;
    } else if(executableUnit.kind === 'clear') {
        return executableUnit;
    } else if(executableUnit.kind === 'drop') {
        return executableUnit;
    } else if(executableUnit.kind === 'create') {
        return executableUnit;
    } else {
        throw new Error('unknown executable unit: ' + executableUnit.kind);
    }
};
AbstractQueryTree.AbstractQueryTree.prototype.parseSelect = function(syntaxTree){
    if(syntaxTree == null) {
        //console.log("error parsing query");
        return null;
    } else {
        var env = { freshCounter: 0 };
        syntaxTree.pattern = this.build(syntaxTree.pattern, env);
        return syntaxTree;
    }
};
AbstractQueryTree.AbstractQueryTree.prototype.parseInsertData = function(syntaxTree){
    if(syntaxTree == null) {
        //console.log("error parsing query");
        return null;
    } else {
        return syntaxTree;
    }
};
AbstractQueryTree.AbstractQueryTree.prototype.build = function(node, env) {
    if(node.token === 'groupgraphpattern') {
        return _buildGroupGraphPattern.call(this, node, env);
    } else if (node.token === 'basicgraphpattern') {
        var bgp = { kind: 'BGP',
            value: node.triplesContext };
        //console.log("pre1");
        bgp = AbstractQueryTree.translatePathExpressionsInBGP(bgp, env);
        //console.log("translation");
        //console.log(sys.inspect(bgp,true,20));
        return bgp;
    } else if (node.token === 'graphunionpattern') {
        var a = this.build(node.value[0],env);
        var b = this.build(node.value[1],env);
        return { kind: 'UNION',
            value: [a,b] };
    } else if(node.token === 'graphgraphpattern') {
        var c = this.build(node.value, env);
        return { kind: 'GRAPH',
```

```
                value: c,
                graph: node.graph };
        } else {
            throw new Error("not supported token in query:"+node.token);
        }
    };
    AbstractQueryTree.translatePathExpressionsInBGP = function(bgp, env) {
        var pathExpression;
        var before = [], rest, bottomJoin;
        for(var i=0; i<bgp.value.length; i++) {
            if(bgp.value[i].predicate && bgp.value[i].predicate.token === 'path') {
                //console.log("FOUND A PATH");
                pathExpression = bgp.value[i];
                rest = bgp.value.slice(i+1);
                var bgpTransformed = AbstractQueryTree.translatePathExpression(pathExpression, env);
                var optionalPattern = null;
                //console.log("BACK FROM TRANSFORMED");
                if(bgpTransformed.kind === 'BGP') {
                    before = before.concat(bgpTransformed.value);
                } else if(bgpTransformed.kind === 'ZERO_OR_MORE_PATH' || bgpTransformed.kind ===
'ONE_OR_MORE_PATH'){
                    //console.log("BEFORE");
                    //console.log(bgpTransformed);
                    if(before.length > 0) {
                        bottomJoin =  {kind: 'JOIN',
                            lvalue: {kind: 'BGP', value:before},
                            rvalue: bgpTransformed};
                    } else {
                        bottomJoin = bgpTransformed;
                    }
                    if(bgpTransformed.kind === 'ZERO_OR_MORE_PATH') {
                        if(bgpTransformed.y.token === 'var' && bgpTransformed.y.value.indexOf
("fresh:")===0 &&
                            bgpTransformed.x.token === 'var' && bgpTransformed.x.value.indexOf
("fresh:")===0) {
                            //console.log("ADDING EXTRA PATTERN 1)");
                            Utils.each(bgp.value, function (value) {
                                //console.log(bgp.value[j]);
                                if(value.object && value.object.token === 'var' && value.object.value ===
bgpTransformed.x.value) {
                                    //console.log(" YES 1)");
                                    optionalPattern = TreeUtils.clone(value);
                                    optionalPattern.object = bgpTransformed.y;
                                }
                            });
                        } else if(bgpTransformed.y.token === 'var' && bgpTransformed.y.value.indexOf
("fresh:")===0) {
                            //console.log("ADDING EXTRA PATTERN 2)");
                            for(var j=0; j<bgp.value.length; j++) {
                                //console.log(bgp.value[j]);
                                if(bgp.value[j].subject && bgp.value[j].subject.token === 'var' &&
bgp.value[j].subject.value === bgpTransformed.y.value) {
                                    //console.log(" YES 2)");
                                    optionalPattern = TreeUtils.clone(bgp.value[j]);
                                    optionalPattern.subject = bgpTransformed.x;
                                }
                            }
                        }
                    }
                    if(rest.length >0) {
                        //console.log("(2a)")
                        var rvalueJoin = AbstractQueryTree.translatePathExpressionsInBGP({kind: 'BGP',
value: rest}, env);
                        //console.log("got rvalue");
                        if(optionalPattern != null) {
                            var optionals = before.concat([optionalPattern]).concat(rest);
                            return { kind: 'UNION',
```

```
                            value: [{ kind: 'JOIN',
                                lvalue: bottomJoin,
                                rvalue: rvalueJoin },
                                {kind: 'BGP',
                                    value: optionals}] };
                    } else {
                        return { kind: 'JOIN',
                            lvalue: bottomJoin,
                            rvalue: rvalueJoin };
                    }
                } else {
                    //console.log("(2b)")
                    return bottomJoin;
                }
            } else {
                // @todo ????
                return bgpTransformed;
            }
        } else {
            before.push(bgp.value[i]);
        }
    }
    //console.log("returning");
    bgp.value = before;
    return bgp;
};
AbstractQueryTree.translatePathExpression  = function(pathExpression, env) {
    var expandedPath;
    // add support for different path patterns
    if(pathExpression.predicate.kind === 'element') {
        // simple paths, maybe modified
        if(pathExpression.predicate.modifier === '+') {
            pathExpression.predicate.modifier = null;
            expandedPath = AbstractQueryTree.translatePathExpression(pathExpression, env);
            return {kind: 'ONE_OR_MORE_PATH',
                path: expandedPath,
                x: pathExpression.subject,
                y: pathExpression.object};
        } else if(pathExpression.predicate.modifier === '*') {
            pathExpression.predicate.modifier = null;
            expandedPath = AbstractQueryTree.translatePathExpression(pathExpression, env);
            return {kind: 'ZERO_OR_MORE_PATH',
                path: expandedPath,
                x: pathExpression.subject,
                y: pathExpression.object};
        } else {
            pathExpression.predicate = pathExpression.predicate.value;
            return {kind: 'BGP', value: [pathExpression]};
        }
    } else if(pathExpression.predicate.kind === 'sequence') {
        var currentSubject = pathExpression.subject;
        var lastObject = pathExpression.object;
        var currentGraph = pathExpression.graph;
        var nextObject, chain;
        var restTriples = [];
        for(var i=0; i< pathExpression.predicate.value.length; i++) {
            if(i!=pathExpression.predicate.value.length-1) {
                nextObject = {
                    token: "var",
                    value: "fresh:"+env.freshCounter
                };
                env.freshCounter++;
            } else {
                nextObject = lastObject;
            }
            // @todo
            // what if the predicate is a path with
```

```
                // '*'? same fresh va in subject and object??
                chain = {
                    subject: currentSubject,
                    predicate: pathExpression.predicate.value[i],
                    object: nextObject
                };
                if(currentGraph != null) {
                    chain.graph = TreeUtils.clone(currentGraph);
                }
                restTriples.push(chain);
                if(i!=pathExpression.predicate.value.length-1) {
                    currentSubject = TreeUtils.clone(nextObject);
                }
            }
            var bgp = {kind: 'BGP', value: restTriples};
            //console.log("BEFORE (1):");
            //console.log(bgp);
            //console.log("--------------");
            return AbstractQueryTree.translatePathExpressionsInBGP(bgp, env);
        }
    };
    function _buildGroupGraphPattern(node, env) {
        var f = (node.filters || []),
            g = {
                kind: "EMPTY_PATTERN"
            },
            parsedPattern,
            that = this;
        Utils.each(node.patterns, function (pattern) {
            if(pattern.token === 'optionalgraphpattern') {
                parsedPattern = that.build(pattern.value, env);
                if(parsedPattern.kind === 'FILTER') {
                    g =  { kind:'LEFT_JOIN',
                        lvalue: g,
                        rvalue: parsedPattern.value,
                        filter: parsedPattern.filter };
                } else {
                    g = { kind:'LEFT_JOIN',
                        lvalue: g,
                        rvalue: parsedPattern,
                        filter: true };
                }
            } else {
                parsedPattern = that.build(pattern,env);
                if(g.kind == "EMPTY_PATTERN") {
                    g = parsedPattern;
                } else {
                    g = { kind: 'JOIN',
                        lvalue: g,
                        rvalue: parsedPattern };
                }
            }
        });
        if(f.length != 0) {
            if(g.kind === 'EMPTY_PATTERN') {
                return { kind: 'FILTER',
                    filter: f,
                    value: g};
            } else if(g.kind === 'LEFT_JOIN' && g.filter === true) {
                return {
                    kind: 'FILTER',
                    filter: f,
                    value: g
                };
            } else if(g.kind === 'LEFT_JOIN') {
                return { kind: 'FILTER',
                    filter: f,
```

```
                value: g};
        } else if(g.kind === 'JOIN') {
            return { kind: 'FILTER',
                filter: f,
                value: g};
        } else if(g.kind === 'UNION') {
            return { kind: 'FILTER',
                filter: f,
                value: g};
        } else if(g.kind === 'GRAPH') {
            return { kind: 'FILTER',
                filter: f,
                value: g};
        } else if(g.kind === 'BGP') {
            return { kind: 'FILTER',
                filter: f,
                value: g};
        } else {
            throw new Error("Unknow kind of algebra expression: "+ g.kind);
        }
    } else {
        return g;
    }
}
/**
 * Collects basic triple pattern in a complex SPARQL AQT
 */
AbstractQueryTree.AbstractQueryTree.prototype.collectBasicTriples = function(aqt, acum) {
    if(acum == null) {
        acum = [];
    }
    if(aqt.kind === 'select') {
        acum = this.collectBasicTriples(aqt.pattern,acum);
    } else if(aqt.kind === 'BGP') {
        acum = acum.concat(aqt.value);
    } else if(aqt.kind === 'ZERO_OR_MORE_PATH') {
        acum = this.collectBasicTriples(aqt.path);
    } else if(aqt.kind === 'UNION') {
        acum = this.collectBasicTriples(aqt.value[0],acum);
        acum = this.collectBasicTriples(aqt.value[1],acum);
    } else if(aqt.kind === 'GRAPH') {
        acum = this.collectBasicTriples(aqt.value,acum);
    } else if(aqt.kind === 'LEFT_JOIN' || aqt.kind === 'JOIN') {
        acum = this.collectBasicTriples(aqt.lvalue, acum);
        acum = this.collectBasicTriples(aqt.rvalue, acum);
    } else if(aqt.kind === 'FILTER') {
        acum = this.collectBasicTriples(aqt.value, acum);
    } else if(aqt.kind === 'construct') {
        acum = this.collectBasicTriples(aqt.pattern,acum);
    } else if(aqt.kind === 'EMPTY_PATTERN') {
        // nothing
    } else {
        throw "Unknown pattern: "+aqt.kind;
    }
    return acum;
};
/**
 * Replaces bindings in an AQT
 */
AbstractQueryTree.AbstractQueryTree.prototype.bind = function(aqt, bindings) {
    if(aqt.graph != null && aqt.graph.token && aqt.graph.token === 'var' &&
        bindings[aqt.graph.value] != null) {
        aqt.graph = bindings[aqt.graph.value];
    }
    if(aqt.filter != null) {
        var acum = [];
        for(var i=0; i< aqt.filter.length; i++) {
```

```
                aqt.filter[i].value = _bindFilter.call(this, aqt.filter[i].value, bindings);
                acum.push(aqt.filter[i]);
            }
            aqt.filter = acum;
        }
        if(aqt.kind === 'select') {
            aqt.pattern = this.bind(aqt.pattern, bindings);
            //acum = this.collectBasicTriples(aqt.pattern,acum);
        } else if(aqt.kind === 'BGP') {
            aqt.value = _bindTripleContext(aqt.value, bindings);
            //acum = acum.concat(aqt.value);
        } else if(aqt.kind === 'ZERO_OR_MORE_PATH') {
            aqt.path = _bindTripleContext(aqt.path, bindings);
            if(aqt.x && aqt.x.token === 'var' && bindings[aqt.x.value] != null) {
                aqt.x = bindings[aqt.x.value];
            }
            if(aqt.y && aqt.y.token === 'var' && bindings[aqt.y.value] != null) {
                aqt.y = bindings[aqt.y.value];
            }
        } else if(aqt.kind === 'UNION') {
            aqt.value[0] = this.bind(aqt.value[0],bindings);
            aqt.value[1] = this.bind(aqt.value[1],bindings);
        } else if(aqt.kind === 'GRAPH') {
            aqt.value = this.bind(aqt.value,bindings);
        } else if(aqt.kind === 'LEFT_JOIN' || aqt.kind === 'JOIN') {
            aqt.lvalue = this.bind(aqt.lvalue, bindings);
            aqt.rvalue = this.bind(aqt.rvalue, bindings);
        } else if(aqt.kind === 'FILTER') {
            aqt.filter = _bindFilter.call(this, aqt.filter[i].value, bindings);
        } else if(aqt.kind === 'EMPTY_PATTERN') {
            // nothing
        } else {
            throw "Unknown pattern: "+aqt.kind;
        }
        return aqt;
    };
    function _bindTripleContext(triples, bindings) {
        Utils.each(triples, function (triple, i) {
            delete triple['graph'];
            delete triple['variables'];
            Utils.each(triple, function (comp, p) {
                if(comp.token === 'var' && bindings[comp.value] != null) {
                    triples[i][p] = bindings[comp.value];
                }
            });
        });
        return triples;
    }
    function _bindFilter(filterExpr, bindings) {
        var that = this;
        if(filterExpr.expressionType != null) {
            var expressionType = filterExpr.expressionType;
            if(expressionType == 'relationalexpression') {
                filterExpr.op1 = _bindFilter.call(that, filterExpr.op1, bindings);
                filterExpr.op2 = _bindFilter.call(that, filterExpr.op2, bindings);
            } else if(expressionType == 'conditionalor' || expressionType == 'conditionaland') {
                Utils.map(filterExpr.operands, function (operand) {
                    return _bindFilter.call(that, operand, bindings);
                });
            } else if(expressionType == 'additiveexpression') {
                filterExpr.summand = _bindFilter.call(that, filterExpr.summand, bindings);
                Utils.each(filterExpr.summands, function (summand) {
                    summand.expression = _bindFilter.call(that, summand.expression, bindings);
                });
            } else if(expressionType == 'builtincall') {
                Utils.map(filterExpr.args, function (arg) {
                    return _bindFilter.call(that, arg, bindings);
```

```
            });
        } else if(expressionType == 'multiplicativeexpression') {
            filterExpr.factor = _bindFilter.call(that, filterExpr.factor, bindings);
            Utils.each(filterExpr.factors, function (factor) {
                factor.expression = _bindFilter.call(that, factor.expression, bindings);
            });
        } else if(expressionType == 'unaryexpression') {
            filterExpr.expression = _bindFilter.call(that, filterExpr.expression, bindings);
        } else if(expressionType == 'irireforfunction') {
            Utils.map(filterExpr.args, function (arg) {
                return _bindFilter.call(that, arg, bindings);
            });
        } else if(expressionType == 'atomic') {
            if(filterExpr.primaryexpression == 'var') {
                // lookup the var in the bindings
                if(bindings[filterExpr.value.value] != null) {
                    var val = bindings[filterExpr.value.value];
                    if(val.token === 'uri') {
                        filterExpr.primaryexpression = 'iri';
                    } else {
                        filterExpr.primaryexpression = 'literal';
                    }
                    filterExpr.value = val;
                }
            }
        }
    }
    return filterExpr;
}
/**
 * Replaces terms in an AQT
 */
AbstractQueryTree.AbstractQueryTree.prototype.replace = function(aqt, from, to, ns) {
    if(aqt.graph != null && aqt.graph.token && aqt.graph.token === from.token &&
        aqt.graph.value == from.value) {
        aqt.graph = TreeUtils.clone(to);
    }
    if(aqt.filter != null) {
        var acum = [];
        for(var i=0; i< aqt.filter.length; i++) {
            aqt.filter[i].value = _replaceFilter(aqt.filter[i].value, from, to, ns);
            acum.push(aqt.filter[i]);
        }
        aqt.filter = acum;
    }
    if(aqt.kind === 'select') {
        aqt.pattern = this.replace(aqt.pattern, from, to, ns);
    } else if(aqt.kind === 'BGP') {
        aqt.value = _replaceTripleContext(aqt.value, from, to, ns);
    } else if(aqt.kind === 'ZERO_OR_MORE_PATH') {
        aqt.path = _replaceTripleContext(aqt.path, from,to, ns);
        if(aqt.x && aqt.x.token === from.token && aqt.value === from.value) {
            aqt.x = TreeUtils.clone(to);
        }
        if(aqt.y && aqt.y.token === from.token && aqt.value === from.value) {
            aqt.y = TreeUtils.clone(to);
        }
    } else if(aqt.kind === 'UNION') {
        aqt.value[0] = this.replace(aqt.value[0],from,to, ns);
        aqt.value[1] = this.replace(aqt.value[1],from,to, ns);
    } else if(aqt.kind === 'GRAPH') {
        aqt.value = this.replace(aqt.value,from,to);
    } else if(aqt.kind === 'LEFT_JOIN' || aqt.kind === 'JOIN') {
        aqt.lvalue = this.replace(aqt.lvalue, from, to, ns);
        aqt.rvalue = this.replace(aqt.rvalue, from, to, ns);
    } else if(aqt.kind === 'FILTER') {
        aqt.value = _replaceFilter(aqt.value, from,to, ns);
```

```
        } else if(aqt.kind === 'EMPTY_PATTERN') {
            // nothing
        } else {
            throw "Unknown pattern: "+aqt.kind;
        }
        return aqt;
    };
    function _replaceTripleContext(triples, from, to, ns) {
        Utils.each(triples, function (triple, i) {
            Utils.each(triple, function (comp, p) {
                if(comp.token === 'var' && from.token === 'var' && comp.value === from.value) {
                    triples[i][p] = to;
                } else if(comp.token === 'blank' && from.token === 'blank' && comp.value === from.value) {
                    triples[i][p] = to;
                } else {
                    if((comp.token === 'literal' || comp.token ==='uri') &&
                        (from.token === 'literal' || from.token ==='uri') &&
                        comp.token === from.token && TreeUtils.lexicalFormTerm(comp,ns)[comp.token] ===
TreeUtils.lexicalFormTerm(from,ns)[comp.token]) {
                        triples[i][p] = to;
                    }
                }
            });
        });
        return triples;
    }
    function _replaceFilter(filterExpr, from, to, ns) {
        if(filterExpr.expressionType != null) {
            var expressionType = filterExpr.expressionType;
            if(expressionType == 'relationalexpression') {
                filterExpr.op1 = _replaceFilter(filterExpr.op1, from, to, ns);
                filterExpr.op2 = _replaceFilter(filterExpr.op2, from, to, ns);
            } else if(expressionType == 'conditionalor' || expressionType == 'conditionaland') {
                Utils.map(filterExpr.operands, function (operand) {
                    return _replaceFilter(operand, from, to, ns);
                });
            } else if(expressionType == 'additiveexpression') {
                filterExpr.summand = _replaceFilter(filterExpr.summand, from, to, ns);
                Utils.each(filterExpr.summands, function (summand) {
                    summand.expression = _replaceFilter(summand.expression, from, to, ns);
                });
            } else if(expressionType == 'builtincall') {
                Utils.map(filterExpr.args, function (arg) {
                    return _replaceFilter(arg, from, to, ns);
                });
            } else if(expressionType == 'multiplicativeexpression') {
                filterExpr.factor = _replaceFilter(filterExpr.factor, from, to, ns);
                Utils.each(filterExpr.factors, function (factor) {
                    factor.expression = _replaceFilter(factor.expression, from, to, ns);
                });
            } else if(expressionType == 'unaryexpression') {
                filterExpr.expression = _replaceFilter(filterExpr.expression, from, to, ns);
            } else if(expressionType == 'irireforfunction') {
                Utils.map(filterExpr.factors.args, function (arg) {
                    return _replaceFilter(arg, from, to, ns);
                });
            } else if(expressionType == 'atomic') {
                var val = null;
                if(filterExpr.primaryexpression == from.token && filterExpr.value == from.value) {
                    val = to.value;
                } else if(filterExpr.primaryexpression == 'iri' && from.token == 'uri' &&
filterExpr.value == from.value) {
                    val = to.value;
                }

                if(val != null) {
                    if(to.token === 'uri') {
```

```
                        filterExpr.primaryexpression = 'iri';
                    } else {
                        filterExpr.primaryexpression = to.token;
                    }
                    filterExpr.value = val;
                }
            }
        }
        return filterExpr;
    }
    AbstractQueryTree.AbstractQueryTree.prototype.treeWithUnion = function(aqt) {
        if(aqt == null)
            return false;
        if(aqt.kind == null)
            return false;
        if(aqt.kind === 'select') {
            return this.treeWithUnion(aqt.pattern);
        } else if(aqt.kind === 'BGP') {
            return this.treeWithUnion(aqt.value);
        } else if(aqt.kind === 'ZERO_OR_MORE_PATH') {
            return false;
        } else if(aqt.kind === 'UNION') {
            if(aqt.value[0].value != null && aqt.value[0].value.variables != null &&
                aqt.value[1].value != null && aqt.value[1].value.variables != null) {
                //console.log("COMPARING:"+aqt.value[0].variables.join("/"));
                //console.log("VS "+aqt.values[1].variables.join("/"));
                if(aqt.value[0].variables.join("/") === aqt.values[1].variables.join("/")) {
                    if(this.treeWithUnion(aqt.value[0]))
                        return true;
                    else
                        return this.treeWithUnion(aqt.value[1]);
                }
            } else {
                return true;
            }
        } else if(aqt.kind === 'GRAPH') {
            return false;
        } else if(aqt.kind === 'LEFT_JOIN' || aqt.kind === 'JOIN') {
            var leftUnion  = this.treeWithUnion(aqt.lvalue);
            if(leftUnion)
                return true;
            else
                this.treeWithUnion(aqt.rvalue);
        } else if(aqt.kind === 'FILTER') {
            return false;
        } else if(aqt.kind === 'EMPTY_PATTERN') {
            return false;
        } else {
            return false;
        }
    };
    return AbstractQueryTree;
});
define([
    "./abstract_query_tree",
    "../rdf-persistence/quad_index_common",
    "./rdf_js_interface",
    "../trees/utils"
], function (AbstractQueryTree, QuadIndexCommon, RDFJSInterface, TreeUtils) {
    var Callbacks = {};
    Callbacks.ANYTHING = {'token': 'var',
        'value': '_'};

    Callbacks.added = 'added';
    Callbacks.deleted = 'deleted';
    Callbacks.eventsFlushed = 'eventsFlushed';
```

```
    Callbacks.CallbacksBackend = function() {
        this.aqt = new AbstractQueryTree.AbstractQueryTree();
        this.engine = arguments[0];
        this.indexMap = {};
        this.observersMap = {};
        this.queriesIndexMap = {};
        this.emptyNotificationsMap = {};
        this.queriesList = [];
        this.pendingQueries = [];
        this.matchedQueries = [];
        this.updateInProgress = null;
        this.indices = ['SPOG', 'GP', 'OGS', 'POG', 'GSP', 'OS'];
        this.componentOrders = {
            SPOG: ['subject', 'predicate', 'object', 'graph'],
            GP: ['graph', 'predicate', 'subject', 'object'],
            OGS: ['object', 'graph', 'subject', 'predicate'],
            POG: ['predicate', 'object', 'graph', 'subject'],
            GSP: ['graph', 'subject', 'predicate', 'object'],
            OS: ['object', 'subject', 'predicate', 'graph']
        };

        this.callbackCounter = 0;
        this.callbacksMap = {};
        this.callbacksInverseMap = {};

        this.queryCounter = 0;
        this.queriesMap = {};
        this.queriesCallbacksMap = {};
        this.queriesInverseMap = {};

        for(var i=0; i<this.indices.length; i++) {
            var indexKey = this.indices[i];
            this.indexMap[indexKey] = {};
            this.queriesIndexMap[indexKey] = {};
        };
    };

    Callbacks.CallbacksBackend.prototype.startGraphModification = function() {
        this.pendingQueries = [].concat(this.queriesList);
        this.matchedQueries = [];

        var added = Callbacks['added'];
        var deleted = Callbacks['deleted'];
        if(this.updateInProgress == null) {
            this.updateInProgress = {added: [], deleted: []};
        }
    };

    Callbacks.CallbacksBackend.prototype.nextGraphModification = function(event, quad) {
        this.updateInProgress[event].push(quad);
    };

    Callbacks.CallbacksBackend.prototype.endGraphModification = function(callback) {
        var that = this;
        if(this.updateInProgress != null) {
            var tmp = that.updateInProgress;
            that.updateInProgress = null;
            this.sendNotification(Callbacks['deleted'], tmp[Callbacks['deleted']],function(){
                that.sendNotification(Callbacks['added'], tmp[Callbacks['added']], function(){
                    that.sendEmptyNotification(Callbacks['eventsFlushed'], null, function(){
                        that.dispatchQueries(function(){
                            callback(true);
                        });
                    });
                });
            });
        } else {
```

```
            callback(true);
        }
    };

    Callbacks.CallbacksBackend.prototype.cancelGraphModification = function() {
        this.updateInProgress = null;
    };

    Callbacks.CallbacksBackend.prototype.sendNotification = function(event, quadsPairs, doneCallback) {
        var notificationsMap = {};
        for(var i=0; i<quadsPairs.length; i++) {
            var quadPair = quadsPairs[i];
            for(var indexKey in this.indexMap) {
                var index = this.indexMap[indexKey];
                var order = this.componentOrders[indexKey];
                this._searchCallbacksInIndex(index, order, event, quadPair, notificationsMap);
                if(this.pendingQueries.length != 0) {
                    index = this.queriesIndexMap[indexKey];
                    this._searchQueriesInIndex(index, order, quadPair);
                }
            }
        }

        this.dispatchNotifications(notificationsMap);

        if(doneCallback != null)
            doneCallback(true);
    };

    Callbacks.CallbacksBackend.prototype.sendEmptyNotification = function(event, value, doneCallback) {
        var callbacks = this.emptyNotificationsMap[event] || [];
        for(var i=0; i<callbacks.length; i++) {
            callbacks[i](event, value);
        }
        doneCallback();
    };

    Callbacks.CallbacksBackend.prototype.dispatchNotifications = function(notificationsMap) {
        for(var callbackId in notificationsMap) {
            var callback = this.callbacksMap[callbackId];
            var deleted = notificationsMap[callbackId][Callbacks['deleted']];
            if(deleted!=null) {
                try {
                    callback(Callbacks['deleted'],deleted);
                }catch(e){}
            }
            for(var event in notificationsMap[callbackId]) {
                if(event!=Callbacks['deleted']) {
                    try{
                        callback(event, notificationsMap[callbackId][event]);
                    }catch(e){}

                }
            }
        }
    };

    Callbacks.CallbacksBackend.prototype._searchCallbacksInIndex = function(index, order, event,
quadPair, notificationsMap) {
        var quadPairNomalized = quadPair[1];
        var quadPair = quadPair[0];

        for(var i=0; i<(order.length+1); i++) {
            var matched = index['_'] || [];

            var filteredIds = [];
            for(var j=0; j<matched.length; j++) {
```

```
            var callbackId = matched[j];
            if(this.callbacksMap[callbackId] != null) {
                notificationsMap[callbackId] = notificationsMap[callbackId] || {};
                notificationsMap[callbackId][event] = notificationsMap[callbackId][event] || [];
                notificationsMap[callbackId][event].push(quadPair);
                filteredIds.push(callbackId);
            }
        }
        index['_'] = filteredIds;
        var component = order[i];
        if(index[''+quadPairNomalized[component]] != null) {
            index = index[''+quadPairNomalized[component]];
        } else {
            break;
        }
    }
};

Callbacks.CallbacksBackend.prototype.subscribeEmpty = function(event, callback) {
    var callbacks = this.emptyNotificationsMap[event] || [];
    callbacks.push(callback);
    this.emptyNotificationsMap[event] = callbacks;
};

Callbacks.CallbacksBackend.prototype.unsubscribeEmpty = function(event, callback) {
    var callbacks = this.emptyNotificationsMap[event];
    if(callbacks != null) {
        callbacks = TreeUtils.remove(callbacks, callback);
    }
    this.emptyNotificationsMap[event] = callbacks;
};

Callbacks.CallbacksBackend.prototype.subscribe = function(s,p,o,g,callback, doneCallback) {
    var quad = this._tokenizeComponents(s,p,o,g);
    var queryEnv = {blanks:{}, outCache:{}};
    this.engine.registerNsInEnvironment(null, queryEnv);
    var that = this;
    var normalized = this.engine.normalizeQuad(quad, queryEnv, true);
    var pattern =  new QuadIndexCommon.Pattern(normalized);
    var indexKey = that._indexForPattern(pattern);
    var indexOrder = that.componentOrders[indexKey];
    var index = that.indexMap[indexKey];
    for(var i=0; i<indexOrder.length; i++) {
        var component = indexOrder[i];
        var quadValue = normalized[component];
        if(quadValue === '_') {
            if(index['_'] == null) {
                index['_'] = [];
            }
            that.callbackCounter++;
            index['_'].push(that.callbackCounter);
            that.callbacksMap[that.callbackCounter] = callback;
            that.callbacksInverseMap[callback] = that.callbackCounter;
            break;
        } else {
            if(i===indexOrder.length-1) {
                index[quadValue] = index[quadValue] || {'_':[]};
                that.callbackCounter++;
                index[quadValue]['_'].push(that.callbackCounter);
                that.callbacksMap[that.callbackCounter] = callback;
                that.callbacksInverseMap[callback] = that.callbackCounter;
            } else {
                index[quadValue] = index[quadValue] || {};
                index = index[quadValue];
            }
        }
    }
```

```
    if(doneCallback != null)
        doneCallback(true);
};

Callbacks.CallbacksBackend.prototype.unsubscribe = function(callback) {
    var id = this.callbacksInverseMap[callback];
    if(id != null) {
        delete this.callbacksInverseMap[callback];
        delete this.callbacksMap[id];
    }
};

Callbacks.CallbacksBackend.prototype._tokenizeComponents = function(s, p, o, g) {
    var pattern = {};

    if(s == null) {
        pattern['subject'] = Callbacks.ANYTHING;
    } else {
        if(s.indexOf("_:") == 0) {
            pattern['subject'] = {'token': 'blank', 'value':s};
        } else {
            pattern['subject'] = {'token': 'uri', 'value':s};
        }
    }

    if(p == null) {
        pattern['predicate'] = Callbacks.ANYTHING;
    } else {
        pattern['predicate'] = {'token': 'uri', 'value':p};
    }

    if(o == null) {
        pattern['object'] = Callbacks.ANYTHING;
    } else {
        pattern['object'] = {'token': 'uri', 'value':o};
    }

    if(g == null) {
        pattern['graph'] = Callbacks.ANYTHING;
    } else {
        pattern['graph'] = {'token': 'uri', 'value':g};
    }

    return pattern;
};

Callbacks.CallbacksBackend.prototype._indexForPattern = function(pattern) {
    var indexKey = pattern.indexKey;
    var matchingIndices = this.indices;

    for(var i=0; i<matchingIndices.length; i++) {
        var index = matchingIndices[i];
        var indexComponents = this.componentOrders[index];
        for(var j=0; j<indexComponents.length; j++) {
            if(TreeUtils.include(indexKey, indexComponents[j])===false) {
                break;
            }
            if(j==indexKey.length-1) {
                return index;
            }
        }
    }

    return 'SPOG'; // If no other match, we return the most generic index
};

Callbacks.CallbacksBackend.prototype.observeNode = function() {
```

```
        var uri,graphUri,callback,doneCallback;

        if(arguments.length === 4) {
            uri = arguments[0];
            graphUri = arguments[1];
            callback = arguments[2];
            doneCallback = arguments[3];
        } else {
            uri = arguments[0];
            graphUri = this.engine.lexicon.defaultGraphUri;
            callback = arguments[1];
            doneCallback = arguments[2];
        }
        var query = "CONSTRUCT { <" + uri + "> ?p ?o } WHERE { GRAPH <" + graphUri + "> { <" + uri + "> ?
p ?o } }";
        var that = this;
        var queryEnv = {blanks:{}, outCache:{}};
        this.engine.registerNsInEnvironment(null, queryEnv);
        var bindings = [];
        this.engine.execute(query,  function(success, graph){
            if(success) {
                var node = graph;
                var mustFlush = false;
                var observer = function(event, triples){
                    if(event === 'eventsFlushed' && mustFlush ) {
                        mustFlush = false;
                        try {
                            callback(node);
                        }catch(e){}
                    } else if(event !== 'eventsFlushed') {
                        mustFlush = true;
                        for(var i = 0; i<triples.length; i++) {
                            var triple = triples[i];
                            var s = RDFJSInterface.buildRDFResource
(triple.subject,bindings,that.engine,queryEnv);
                            var p = RDFJSInterface.buildRDFResource
(triple.predicate,bindings,that.engine,queryEnv);
                            var o = RDFJSInterface.buildRDFResource
(triple.object,bindings,that.engine,queryEnv);
                            if(s!=null && p!=null && o!=null) {
                                triple = new RDFJSInterface.Triple(s,p,o);
                                if(event === Callbacks['added']) {
                                    node.add(triple);
                                } else if(event === Callbacks['deleted']) {
                                    node.remove(triple);
                                }
                            }
                        }
                    }
                };
                that.observersMap[callback] = observer;
                that.subscribeEmpty(Callbacks['eventsFlushed'], observer);
                that.subscribe(uri,null,null,null,observer,function(){
                    try {
                        callback(node);
                    }catch(e){}

                    if(doneCallback)
                        doneCallback(true)
                });
            } else {
                if(doneCallback)
                    doneCallback(false);
            }
        });
    };
```

```
    Callbacks.CallbacksBackend.prototype.stopObservingNode = function(callback) {
        var observer = this.observersMap[callback];
        if(observer) {
            this.unsubscribe(observer);
            this.unsubscribeEmpty(Callbacks['eventsFlushed'],observer);
            return true;
        } else {
            return false;
        }
    };

// Queries

    Callbacks.CallbacksBackend.prototype.observeQuery = function(query, callback, endCallback) {
        var queryParsed = this.aqt.parseQueryString(query);
        var parsedTree = this.aqt.parseSelect(queryParsed.units[0]);
        var patterns = this.aqt.collectBasicTriples(parsedTree);
        var that = this;
        var queryEnv = {blanks:{}, outCache:{}};
        this.engine.registerNsInEnvironment(null, queryEnv);
        var floop, pattern, quad, indexKey, indexOrder, index;

        var counter = this.queryCounter;
        this.queryCounter++;
        this.queriesMap[counter] = query;
        this.queriesInverseMap[query] = counter;
        this.queriesList.push(counter);
        this.queriesCallbacksMap[counter] = callback;

        for(var i=0; i<patterns.length; i++) {
            quad = patterns[i];
            if(quad.graph == null) {
                quad.graph = that.engine.lexicon.defaultGraphUriTerm;
            }

            var normalized = that.engine.normalizeQuad(quad, queryEnv, true);
            pattern =  new QuadIndexCommon.Pattern(normalized);
            indexKey = that._indexForPattern(pattern);
            indexOrder = that.componentOrders[indexKey];
            index = that.queriesIndexMap[indexKey];

            for(var j=0; j<indexOrder.length; j++) {
                var component = indexOrder[j];
                var quadValue = normalized[component];
                if(typeof(quadValue) === 'string') {
                    if(index['_'] == null) {
                        index['_'] = [];
                    }
                    index['_'].push(counter);
                    break;
                } else {
                    if(j===indexOrder.length-1) {
                        index[quadValue] = index[quadValue] || {'_':[]};
                        index[quadValue]['_'].push(counter);
                    } else {
                        index[quadValue] = index[quadValue] || {};
                        index = index[quadValue];
                    }
                }
            }

        }

        this.engine.execute(query, function(success, results){
            if(success){
                callback(results);
            } else {
```

```
            //console.log("ERROR in query callback "+results);
        }
    });

    if(endCallback != null)
        endCallback();
};

Callbacks.CallbacksBackend.prototype.stopObservingQuery = function(query) {
    var id = this.queriesInverseMap[query];
    if(id != null) {
        delete this.queriesInverseMap[query];
        delete this.queriesMap[id];
        this.queriesList = TreeUtils.remove(this.queriesList, id);
    }
};

Callbacks.CallbacksBackend.prototype._searchQueriesInIndex = function(index, order, quadPair) {
    var quadPairNomalized = quadPair[1];
    var quadPair = quadPair[0];

    for(var i=0; i<(order.length+1); i++) {
        var matched = index['_'] || [];

        var filteredIds = [];
        for(var j=0; j<matched.length; j++) {
            var queryId = matched[j];
            if(TreeUtils.include(this.pendingQueries,queryId)) {
                TreeUtils.remove(this.pendingQueries,queryId);
                this.matchedQueries.push(queryId);
            }
            // removing IDs for queries no longer being observed
            if(this.queriesMap[queryId] != null) {
                filteredIds.push(queryId);
            }
        }
        index['_'] = filteredIds;

        var component = order[i];
        if(index[''+quadPairNomalized[component]] != null) {
            index = index[''+quadPairNomalized[component]];
        } else {
            break;
        }
    }
};

Callbacks.CallbacksBackend.prototype.dispatchQueries = function(callback) {
    var that = this;
    var floop, query, queryId, queryCallback;
    var toDispatchMap = {};

    TreeUtils.repeat(0, this.matchedQueries.length,
        function(k, env){
            floop = arguments.callee;
            queryId = that.matchedQueries[env._i];
            // avoid duplicate notifications
            if(toDispatchMap[queryId] == null) {
                toDispatchMap[queryId] = true;
                query = that.queriesMap[queryId];
                queryCallback = that.queriesCallbacksMap[queryId];
                TreeUtils.recur(function(){
                    that.engine.execute(query,
                        function(success, results){
                            if(success) {
                                try{
                                    queryCallback(results);
```

```
                                    }catch(e){}
                                }
                                k(floop,env);
                            });
                    });
                } else {
                    k(floop,env);
                }
            },
            function(env) {
                callback();
            });
    };
    return Callbacks;
});define([
    "./abstract_query_tree",
    "./../trees/utils",
    "./../rdf-persistence/quad_index_common",
    "./query_plan_sync_dpsize",
    "./query_filters",
    "./rdf_js_interface",
    "./../communication/rdf_loader",
    "./callbacks",
    "./../../graphite/utils"
], function (AbstractQueryTree, TreeUtils, QuadIndexCommon, QueryPlan, QueryFilters, RDFJSInterface,
RDFLoader, Callbacks, Utils) {
    var QueryEngine = {};
    QueryEngine.QueryEngine = function(params) {
        if(arguments.length != 0) {
            this.backend = params.backend;
            this.lexicon = params.lexicon;
            // batch loads should generate events?
            this.eventsOnBatchLoad = (params.eventsOnBatchLoad || false);
            // list of namespaces that will be automatically added to every query
            this.defaultPrefixes = {};
            this.abstractQueryTree = new AbstractQueryTree.AbstractQueryTree();
            this.rdfLoader = new RDFLoader.RDFLoader();
            //this.rdfLoader = new RDFLoader.RDFLoader(params['communication']);
            this.callbacksBackend = new Callbacks.CallbacksBackend(this);
        }
    };
    // Utils
    QueryEngine.QueryEngine.prototype.registerNsInEnvironment = function(prologue, env) {
        var prefixes = [],
            toSave = {},
            p,
            i;
        if(prologue != null && prologue.prefixes != null) {
            prefixes =prologue.prefixes;
        }
        // adding default prefixes;
        for(p in this.defaultPrefixes) {
            if (this.defaultPrefixes.hasOwnProperty(p)) {
                toSave[p] = this.defaultPrefixes[p];
            }
        }
        for(i=0; i<prefixes.length; i++) {
            var prefix = prefixes[i];
            if(prefix.token === "prefix") {
                toSave[prefix.prefix] = prefix.local;
            }
        }
        env.namespaces = toSave;
        if(prologue!=null && prologue.base && typeof(prologue.base) === 'object') {
            env.base = prologue.base.value;
        } else {
            env.base = null;
```

```
    }
};
QueryEngine.QueryEngine.prototype.applyModifier = function(modifier, projectedBindings) {
    var map,
        result,
        bindings,
        key,
        i,
        p,
        obj;
    if(modifier == "DISTINCT") {
        map = {};
        result = [];
        for(i=0; i<projectedBindings.length; i++) {
            bindings = projectedBindings[i];
            key = "";
            // if no projection variables hash is passed, all the bound
            // variable in the current bindings will be used.
            for(p in (bindings)) {
                if (bindings.hasOwnProperty(p)) {
                    // hashing the object
                    obj = bindings[p];
                    if(obj == null) {
                        key = key+p+'null';
                    } else if(obj.token == 'literal') {
                        if(obj.value != null) {
                            key = key + obj.value;
                        }
                        if(obj.lang != null) {
                            key = key + obj.lang;
                        }
                        if(obj.type != null) {
                            key = key + obj.type;
                        }
                    } else if(obj.value) {
                        key  = key + p + obj.value;
                    } else {
                        key = key + p + obj;
                    }
                }
            }
            if(map[key] == null) {
                // this will preserve the order in projectedBindings
                result.push(bindings);
                map[key] = true;
            }
        }
        return result;
    } else {
        return projectedBindings;
    }
};

QueryEngine.QueryEngine.prototype.applyLimitOffset = function(offset, limit, bindings) {
    if(limit == null && offset == null) {
        return bindings;
    }

    if (offset == null) {
        offset = 0;
    }

    if(limit == null) {
        limit = bindings.length;
    } else {
        limit = offset + limit;
    }
```

```
        return bindings.slice(offset, limit);
    };


    QueryEngine.QueryEngine.prototype.applySingleOrderBy = function(orderFilters, modifiedBindings,
dataset, outEnv) {
        var acum = [];
        for(var i=0; i<orderFilters.length; i++) {
            var orderFilter = orderFilters[i];
            var results = QueryFilters.collect(orderFilter.expression, [modifiedBindings], dataset,
outEnv, this);
            acum.push(results[0].value);
        }
        return {binding:modifiedBindings, value:acum};
    };

    QueryEngine.QueryEngine.prototype.applyOrderBy = function(order, modifiedBindings, dataset, outEnv) {
        var that = this,
            acum = [],
            i;
        if(order != null && order.length > 0) {
            for(i=0; i<modifiedBindings.length; i++) {
                var bindings = modifiedBindings[i];
                var results = that.applySingleOrderBy(order, bindings, dataset, outEnv);
                acum.push(results);
            }
            acum.sort(function(a,b){
                return that.compareFilteredBindings(a, b, order, outEnv);
            });
            var toReturn = [];
            for(i=0; i<acum.length; i++) {
                toReturn.push(acum[i].binding);
            }
            return toReturn;
        } else {
            return modifiedBindings;
        }
    };

    QueryEngine.QueryEngine.prototype.compareFilteredBindings = function(a, b, order, env) {
        var found = false;
        var i = 0;
        while(!found) {
            if(i==a.value.length) {
                return 0;
            }
            var direction = order[i].direction;
            var filterResult;

            // unbound first
            if(a.value[i] == null && b.value[i] == null) {
                i++;
                continue;
            }else if(a.value[i] == null) {
                filterResult = {value: false};
            } else if(b.value[i] == null) {
                filterResult = {value: true};
            } else
            // blanks
            if(a.value[i].token === 'blank' && b.value[i].token === 'blank') {
                i++;
                continue;
            } else if(a.value[i].token === 'blank') {
                filterResult = {value: false};
            } else if(b.value[i].token === 'blank') {
                filterResult = {value: true};
```

```
            } else
            // uris
            if(a.value[i].token === 'uri' && b.value[i].token === 'uri') {
                if(QueryFilters.runEqualityFunction(a.value[i], b.value[i], [], this, env).value == true)
{
                    i++;
                    continue;
                } else {
                    filterResult = QueryFilters.runTotalGtFunction(a.value[i], b.value[i]);
                }
            } else if(a.value[i].token === 'uri') {
                filterResult = {value: false};
            } else if(b.value[i].token === 'uri') {
                filterResult = {value: true};
            } else
            // simple literals
            if(a.value[i].token === 'literal' && b.value[i].token === 'literal' && a.value[i].type ==
null && b.value[i].type == null) {
                if(QueryFilters.runEqualityFunction(a.value[i], b.value[i], [], this, env).value == true)
{
                    i++;
                    continue;
                } else {
                    filterResult = QueryFilters.runTotalGtFunction(a.value[i], b.value[i]);
                }
            } else if(a.value[i].token === 'literal' && a.value[i].type == null) {
                filterResult = {value: false};
            } else if(b.value[i].token === 'literal' && b.value[i].type == null) {
                filterResult = {value: true};
            } else
            // literals
            if(QueryFilters.runEqualityFunction(a.value[i], b.value[i], [], this, env).value == true) {
                i++;
                continue;
            } else {
                filterResult = QueryFilters.runTotalGtFunction(a.value[i], b.value[i]);
            }
            // choose value for comparison based on the direction
            if(filterResult.value == true) {
                if(direction === "ASC") {
                    return 1;
                } else {
                    return -1;
                }
            } else {
                if(direction === "ASC") {
                    return -1;
                } else {
                    return 1;
                }
            }
        }
    };
    QueryEngine.QueryEngine.prototype.removeDefaultGraphBindings = function(bindingsList, dataset) {
        var onlyDefaultDatasets = [];
        var namedDatasetsMap = {};
        for(var i=0; i<dataset.named.length; i++) {
            namedDatasetsMap[dataset.named[i].oid] = true;
        }
        for(i=0; i<dataset.implicit.length; i++) {
            if(namedDatasetsMap[dataset.implicit[i].oid] == null) {
                onlyDefaultDatasets.push(dataset.implicit[i].oid);
            }
        }
        var acum = [];
        for(i=0; i<bindingsList.length; i++) {
            var bindings = bindingsList[i];
```

```
            var foundDefaultGraph = false;
            for(var p in bindings) {
                if (bindings.hasOwnProperty(p)) {
                    for(var j=0; j<namedDatasetsMap.length; j++) {
                        if(bindings[p] === namedDatasetsMap[j]) {
                            foundDefaultGraph = true;
                            break;
                        }
                    }
                    if(foundDefaultGraph) {
                        break;
                    }
                }
            }
            if(!foundDefaultGraph) {
                acum.push(bindings);
            }
        }
        return acum;
    };
    QueryEngine.QueryEngine.prototype.aggregateBindings = function(projection, bindingsGroup, dataset,
env) {
        var denormBindings = this.copyDenormalizedBindings(bindingsGroup, env.outCache);
        var aggregatedBindings = {};
        for(var i=0; i<projection.length; i++) {
            var aggregatedValue = QueryFilters.runAggregator(projection[i], denormBindings, this,
dataset, env);
            if(projection[i].alias) {
                aggregatedBindings[projection[i].alias.value] = aggregatedValue;
            } else {
                aggregatedBindings[projection[i].value.value] = aggregatedValue;
            }
        }
        return(aggregatedBindings);
    };
    QueryEngine.QueryEngine.prototype.projectBindings = function(projection, results, dataset) {
        if(projection[0].kind === '*') {
            return results;
        } else {
            var projectedResults = [];

            for(var i=0; i<results.length; i++) {
                var currentResult = results[i];
                var currentProjected = {};
                var shouldAdd = true;

                for(var j=0; j< projection.length; j++) {
                    if(projection[j].token == 'variable' && projection[j].kind != 'aliased') {
                        currentProjected[projection[j].value.value] = currentResult[projection
[j].value.value];
                    } else if(projection[j].token == 'variable' && projection[j].kind == 'aliased') {
                        var ebv = QueryFilters.runFilter(projection[j].expression, currentResult, this,
dataset, {blanks:{}, outCache:{}});
                        if(QueryFilters.isEbvError(ebv)) {
                            shouldAdd = false;
                            break;
                        } else {
                            currentProjected[projection[j].alias.value] = ebv;
                        }
                    }
                }
                if(shouldAdd === true) {
                    projectedResults.push(currentProjected);
                }
            }
            return projectedResults;
        }
```

```
};
QueryEngine.QueryEngine.prototype.resolveNsInEnvironment = function(prefix, env) {
    var namespaces = env.namespaces;
    return namespaces[prefix];
};
QueryEngine.QueryEngine.prototype.termCost = function(term, env) {
    if(term.token === 'uri') {
        var uri = TreeUtils.lexicalFormBaseUri(term, env);
        if(uri == null) {
            return(0);
        } else {
            return(this.lexicon.resolveUriCost(uri));
        }
    } else if(term.token === 'literal') {
        var lexicalFormLiteral = TreeUtils.lexicalFormLiteral(term, env);
        return(this.lexicon.resolveLiteralCost(lexicalFormLiteral));
    } else if(term.token === 'blank') {
        var label = term.value;
        return this.lexicon.resolveBlankCost(label);
    } else if(term.token === 'var') {
        return (this.lexicon.oidCounter/3)
    } else {
        return(null);
    }
};
QueryEngine.QueryEngine.prototype.normalizeTerm = function(term, env, shouldIndex) {
    var uri,
        lexicalFormLiteral,
        oid,
        label;
    if(term.token === 'uri') {
        uri = TreeUtils.lexicalFormBaseUri(term, env);
        if(uri == null) {
            return(null);
        } else {
            if(shouldIndex) {
                return this.lexicon.registerUri(uri);
            } else {
                return this.lexicon.resolveUri(uri);
            }
        }
    } else if(term.token === 'literal') {
        lexicalFormLiteral = TreeUtils.lexicalFormLiteral(term, env);
        if(shouldIndex) {
            return this.lexicon.registerLiteral(lexicalFormLiteral);
        } else {
            return this.lexicon.resolveLiteral(lexicalFormLiteral);
        }
    } else if(term.token === 'blank') {
        label = term.value;
        oid = env.blanks[label];
        if (oid != null) {
            return oid;
        } else {
            if(shouldIndex) {
                oid = this.lexicon.registerBlank(label);
                env.blanks[label] = oid;
                return(oid);
            } else {
                oid = this.lexicon.resolveBlank(label);
                env.blanks[label] = oid;
                return(oid);
            }
        }
    } else if(term.token === 'var') {
        return(term.value);
    } else {
```

```
            return(null);
        }
    };
    QueryEngine.QueryEngine.prototype.normalizeDatasets = function(datasets, outerEnv) {
        var that = this;
        for(var i=0; i<datasets.length; i++) {
            var dataset = datasets[i];
            if(dataset.value === that.lexicon.defaultGraphUri) {
                dataset.oid = that.lexicon.defaultGraphOid;
            } else {
                var oid = that.normalizeTerm(dataset, outerEnv, false);
                if(oid != null) {
                    dataset.oid = oid;
                } else {
                    return(null);
                }
            }
        }
        return true
    };
    QueryEngine.QueryEngine.prototype.normalizeQuad = function(quad, queryEnv, shouldIndex) {
        var subject    = null;
        var predicate  = null;
        var object     = null;
        var graph      = null;
        var oid;
        if(quad.graph == null) {
            graph = 0; // default graph
        } else {
            oid = this.normalizeTerm(quad.graph, queryEnv, shouldIndex);
            if(oid!=null) {
                graph = oid;
                if(shouldIndex === true && quad.graph.token!='var')
                    this.lexicon.registerGraph(oid);
            } else {
                return null;
            }
        }
        oid = this.normalizeTerm(quad.subject, queryEnv, shouldIndex);
        if(oid!=null) {
            subject = oid;
        } else {
            return null
        }
        oid = this.normalizeTerm(quad.predicate, queryEnv, shouldIndex);
        if(oid!=null) {
            predicate = oid;
        } else {
            return null
        }
        oid = this.normalizeTerm(quad.object, queryEnv, shouldIndex);
        if(oid!=null) {
            object = oid;
        } else {
            return null;
        }
        return({
            subject:subject,
            predicate:predicate,
            object:object,
            graph:graph
        });
    };
    QueryEngine.QueryEngine.prototype.quadCost = function(quad, queryEnv) {
        var graph = quad.graph == null
                ? (this.lexicon.oidCounter/4)
                : this.termCost(quad.graph, queryEnv),
```

```
            subject = this.termCost(quad.subject, queryEnv),
            predicate = this.termCost(quad.predicate, queryEnv),
            object = this.termCost(quad.object, queryEnv);
        return graph+subject+predicate+object;
    };
    QueryEngine.QueryEngine.prototype.denormalizeBindingsList = function(bindingsList, env) {
        var results = [],
            i;
        for(i=0; i<bindingsList.length; i++) {
            var result = this.denormalizeBindings(bindingsList[i], env);
            results.push(result);
        }
        return(results);
    };
    /**
     * Receives a bindings map (var -> oid) and an out cache (oid -> value)
     * returns a bindings map (var -> value) storing in cache all the missing values for oids
     *
     * This is required just to save lookups when final results are generated.
     *
     * @param bindingsList
     * @param out
     * @return {Array}
     */
    QueryEngine.QueryEngine.prototype.copyDenormalizedBindings = function(bindingsList, out) {
        var denormList = [];
        for(var i=0; i<bindingsList.length; i++) {
            var denorm = {};
            var bindings = bindingsList[i];
            var variables = TreeUtils.keys(bindings);
            for(var j=0; j<variables.length; j++) {
                var oid = bindings[variables[j]];
                if(oid == null) {
                    // this can be null, e.g. union different variables (check SPARQL recommendation
examples UNION)
                    denorm[variables[j]] = null;
                } else if(typeof(oid) === 'object') {
                    // the binding is already denormalized, this can happen for example because the value
of the
                    // binding is the result of the aggregation of other bindings in a GROUP clause
                    denorm[variables[j]] = oid;
                } else {
                    var inOut = out[oid];
                    if(inOut!= null) {
                        denorm[variables[j]] = inOut;
                    } else {
                        var val = this.lexicon.retrieve(oid);
                        out[oid] = val;
                        denorm[variables[j]] = val;
                    }
                }
            }
            denormList.push(denorm);
        }
        return denormList;
    };
    QueryEngine.QueryEngine.prototype.denormalizeBindings = function(bindings, env) {
        var variables = TreeUtils.keys(bindings),
            envOut = env.outCache,
            oid,
            val;
        for(var i=0; i<variables.length; i++) {
            oid = bindings[variables[i]];
            if(oid == null) {
                // this can be null, e.g. union different variables (check SPARQL recommendation examples
UNION)
                bindings[variables[i]] = null;
```

```
            } else {
                if(envOut[oid] != null) {
                    bindings[variables[i]] = envOut[oid];
                } else {
                    val = this.lexicon.retrieve(oid);
                    bindings[variables[i]] = val;
                    if(val.token === 'blank') {
                        env.blanks[val.value] = oid;
                    }
                }
            }
        }
        return bindings;
    };
    /**
     * Queries execution
     * @param query
     * @param callback
     * @param [defaultDataset]
     * @param [namedDataset]
     */
    QueryEngine.QueryEngine.prototype.execute = function(query, callback, defaultDataset, namedDataset) {
        var syntaxTree;
        if (Utils.isString(query)) {
            //console.log("IN QUERY ENGINE, QUERY ISN'T PARSED ALREADY", query);
            query = TreeUtils.normalizeUnicodeLiterals(query);
            syntaxTree = this.abstractQueryTree.parseQueryString(query);
        } else {
            //console.log("IN QUERY ENGINE, QUERY IS PARSED ALREADY");
            syntaxTree = query;
        }
        if(syntaxTree === null) {
            callback(false,"Error parsing query string");
        } else {
            if(syntaxTree.token === 'query' && syntaxTree.kind === 'update')  {
                this.callbacksBackend.startGraphModification();
                var that = this;
                this.executeUpdate(syntaxTree, function(success, result){
                    if(that.lexicon["updateAfterWrite"]) {
                        that.lexicon["updateAfterWrite"]();
                    }
                    if(success) {
                        that.callbacksBackend.endGraphModification(function(){
                            callback(success, result);
                        });
                    } else {
                        that.callbacksBackend.cancelGraphModification();
                        callback(success, result);
                    }
                });
            } else if(syntaxTree.token === 'query' && syntaxTree.kind == 'query') {
                //console.log("IN QUERY ENGINE, KIND QUERY");
                this.executeQuery(syntaxTree, callback, defaultDataset, namedDataset);
            }
        }
    };
    // Retrieval queries
    QueryEngine.QueryEngine.prototype.executeQuery = function(syntaxTree, callback, defaultDataset,
namedDataset) {
        var prologue = syntaxTree.prologue,
            units = syntaxTree.units,
            that = this,
            queryEnv = {
                blanks: {},
                outCache: {}
            },
            aqt,
```

```
            graph,
            i,
            j,
            s,
            o,
            blankIdCounter,
            toClear = [],
            components,
            component,
            tripleTemplate;
        // environment for the operation -> base ns, declared ns, etc.
        this.registerNsInEnvironment(prologue, queryEnv);
        // retrieval queries can only have 1 executable unit
        aqt = that.abstractQueryTree.parseExecutableUnit(units[0]);
        // can be anything else but a select???
        if(aqt.kind === 'select') {
            this.executeSelect(aqt, queryEnv, defaultDataset, namedDataset, function(success, result){
                if(success) {
                    if(typeof(result) === 'object' && result.denorm === true) {
                        callback(true, result['bindings']);
                    } else {
                        result = that.denormalizeBindingsList(result, queryEnv);
                        if(result != null) {
                            callback(true, result);
                        } else {
                            callback(false, result);
                        }
                    }
                } else {
                    callback(false, result);
                }
            });
        } else if(aqt.kind === 'ask') {
            aqt.projection = [{"token": "variable", "kind": "*"}];
            this.executeSelect(aqt, queryEnv, defaultDataset, namedDataset, function(success, result){
                if(success) {
                    if(success) {
                        if(result.length>0) {
                            callback(true, true);
                        } else {
                            callback(true, false);
                        }
                    } else {
                        callback(false, result);
                    }
                } else {
                    callback(false, result);
                }
            });
        } else if(aqt.kind === 'construct') {
            aqt.projection = [{"token": "variable", "kind": "*"}];
            that = this;
            this.executeSelect(aqt, queryEnv, defaultDataset, namedDataset, function(success, result){
                if(success) {
                    if(success) {
                        result = that.denormalizeBindingsList(result, queryEnv);
                        if(result != null) {
                            graph = new RDFJSInterface.Graph();
                            // CONSTRUCT WHERE {} case
                            if(aqt.template == null) {
                                aqt.template = {triplesContext: aqt.pattern};
                            }
                            blankIdCounter = 1;
                            toClear = [];
                            for(i=0; i<result.length; i++) {
                                var bindings = result[i];
                                for(j=0; j<toClear.length; j++) {
```

```
                                    delete toClear[j].valuetmp;
                                }
                                for(j=0; j<aqt.template.triplesContext.length; j++) {
                                    // fresh IDs for blank nodes in the construct template
                                    components = ['subject', 'predicate', 'object'];
                                    tripleTemplate = aqt.template.triplesContext[j];
                                    for(var p=0; p<components.length; p++) {
                                        component = components[p];
                                        if(tripleTemplate[component].token === 'blank') {
                                            if(tripleTemplate[component].valuetmp && tripleTemplate
[component].valuetmp != null) {
                                            } else {
                                                var blankId = "_:b"+blankIdCounter;
                                                blankIdCounter++;
                                                tripleTemplate[component].valuetmp = blankId;
                                                toClear.push(tripleTemplate[component]);
                                            }
                                        }
                                    }
                                    s = RDFJSInterface.buildRDFResource
(tripleTemplate.subject,bindings,that,queryEnv);
                                    p = RDFJSInterface.buildRDFResource
(tripleTemplate.predicate,bindings,that,queryEnv);
                                    o = RDFJSInterface.buildRDFResource
(tripleTemplate.object,bindings,that,queryEnv);
                                    if(s && p && o) {
                                        var triple = new RDFJSInterface.Triple(s,p,o);
                                        graph.add(triple);
                                        //} else {
                                        //    return callback(false, "Error creating output graph")
                                    }
                                }
                            }
                            callback(true, graph);
                        } else {
                            callback(false, result);
                        }
                    } else {
                        callback(false, result);
                    }
                } else {
                    callback(false, result);
                }
            });
        }
    };
    // Select queries
    QueryEngine.QueryEngine.prototype.executeSelect = function(unit, env, defaultDataset, namedDataset,
callback) {
        var projection,
            dataset,
            modifier,
            limit,
            offset,
            order,
            that = this,
            i;
        if(unit.kind === "select" || unit.kind === "ask" || unit.kind === "construct" || unit.kind ===
"modify") {
            projection = unit.projection;
            dataset    = unit.dataset;
            modifier   = unit.modifier;
            limit      = unit.limit;
            offset     = unit.offset;
            order      = unit.order;
            if(defaultDataset != null || namedDataset != null) {
                dataset.implicit = defaultDataset || [];
```

```
                dataset.named   = namedDataset || [];
            }
            if(dataset.implicit != null && dataset.implicit.length === 0 && dataset.named !=null &&
dataset.named.length === 0) {
                // We add the default graph to the default merged graph
                dataset.implicit.push(this.lexicon.defaultGraphUriTerm);
            }
            if (that.normalizeDatasets(dataset.implicit.concat(dataset.named), env) != null) {
                var result = that.executeSelectUnit(projection, dataset, unit.pattern, env);
                if(result != null) {
                    // detect single group
                    if(unit.group!=null && unit.group === "") {
                        var foundUniqueGroup = false;
                        for(i=0; i<unit.projection.length; i++) {
                            if(unit.projection[i].expression!=null && unit.projection
[i].expression.expressionType === 'aggregate') {
                                foundUniqueGroup = true;
                                break;
                            }
                        }
                        if(foundUniqueGroup === true) {
                            unit.group = 'singleGroup';
                        }
                    }
                    if(unit.group && unit.group != "") {
                        if(that.checkGroupSemantics(unit.group,projection)) {
                            var groupedBindings = that.groupSolution(result, unit.group, dataset, env);
                            var aggregatedBindings = [];
                            for(i=0; i<groupedBindings.length; i++) {
                                var resultingBindings = that.aggregateBindings(projection, groupedBindings
[i], dataset, env);
                                aggregatedBindings.push(resultingBindings);
                            }
                            callback(true, {'bindings': aggregatedBindings, 'denorm':true});
                        } else {
                            callback(false, "Incompatible Group and Projection variables");
                        }
                    } else {
                        var orderedBindings = that.applyOrderBy(order, result, dataset, env);
                        var projectedBindings = that.projectBindings(projection, orderedBindings,
dataset);
                        var modifiedBindings = that.applyModifier(modifier, projectedBindings);
                        var limitedBindings  = that.applyLimitOffset(offset, limit, modifiedBindings);
                        var filteredBindings = that.removeDefaultGraphBindings(limitedBindings, dataset);
                        callback(true, filteredBindings);
                    }
                } else { // fail selectUnit
                    callback(false, result);
                }
            } else { // fail  normalizaing datasets
                callback(false,"Error normalizing datasets");
            }
        } else {
            callback(false,"Cannot execute " + unit.kind + " query as a select query");
        }
    };
    QueryEngine.QueryEngine.prototype.groupSolution = function(bindings, group, dataset, queryEnv){
        var order = [],
            filteredBindings = [],
            initialized = false,
            that = this,
            i,
            j,
            denormBindings,
            filterResultEbv;
        if(group === 'singleGroup') {
            return [bindings];
```

```
        } else {
            for(i=0; i<bindings.length; i++) {
                var currentBindings = bindings[i];
                var mustAddBindings = true;
                /**
                 * In this loop, we iterate through all the group clauses and tranform the current
bindings
                 * according to the group by clauses.
                 * If it is the first iteration we also save in a different array the order for the
                 * grouped variables that will be used later to build the final groups
                 */
                for(j=0; j<group.length; j++) {
                    var currentOrderClause = group[j];
                    var orderVariable = null;
                    if(currentOrderClause.token === 'var') {
                        orderVariable = currentOrderClause.value;
                        if(initialized == false) {
                            order.push(orderVariable);
                        }
                    } else if(currentOrderClause.token === 'aliased_expression') {
                        orderVariable = currentOrderClause.alias.value;
                        if(initialized == false) {
                            order.push(orderVariable);
                        }
                        if(currentOrderClause.expression.primaryexpression === 'var') {
                            currentBindings[currentOrderClause.alias.value] = currentBindings
[currentOrderClause.expression.value.value];
                        } else {
                            denormBindings = this.copyDenormalizedBindings([currentBindings],
queryEnv.outCache);
                            filterResultEbv = QueryFilters.runFilter(currentOrderClause.expression,
denormBindings[0], that, dataset, queryEnv);
                            if(!QueryFilters.isEbvError(filterResultEbv)) {
                                if(filterResultEbv.value != null) {
                                    filterResultEbv.value = ""+filterResultEbv.value;
                                }
                                currentBindings[currentOrderClause.alias.value]= filterResultEbv;
                            } else {
                                mustAddBindings = false;
                            }
                        }
                    } else {
                        // In this case, we create an additional variable in the binding to hold the
group variable value
                        denormBindings = that.copyDenormalizedBindings([currentBindings],
queryEnv.outCache);
                        filterResultEbv = QueryFilters.runFilter(currentOrderClause, denormBindings[0],
that, queryEnv);
                        mustAddBindings = false;
                    }
                }
                if(initialized == false) {
                    initialized = true;
                }
                if(mustAddBindings === true) {
                    filteredBindings.push(currentBindings);
                }
            }
            /**
             * After processing all the bindings, we build the group using the
             * information stored about the order of the group variables.
             */
            var dups = {};
            var groupMap = {};
            var groupCounter = 0;
            for(i=0; i<filteredBindings.length; i++) {
                var currentTransformedBinding = filteredBindings[i];
```

```
            var key = "";
            for(j=0; j<order.length; j++) {
                var maybeObject = currentTransformedBinding[order[j]];
                if(typeof(maybeObject) === 'object') {
                    key = key + maybeObject.value;
                } else {
                    key = key + maybeObject;
                }
            }
            if(dups[key] == null) {
                //currentTransformedBinding["__group__"] = groupCounter;
                groupMap[key] = groupCounter;
                dups[key] = [currentTransformedBinding];
                //groupCounter++
            } else {
                //currentTransformedBinding["__group__"] = dups[key][0]["__group__"];
                dups[key].push(currentTransformedBinding);
            }
        }
        // The final result is an array of arrays with all the groups
        var groups = [];
        for(var k in dups) {
            if (dups.hasOwnProperty(k)) {
                groups.push(dups[k]);
            }
        }
        return groups;
    }
};
/**
 * Here, all the constructions of the SPARQL algebra are handled
 */
QueryEngine.QueryEngine.prototype.executeSelectUnit = function(projection, dataset, pattern, env) {
    if(pattern.kind === "BGP") {
        return this.executeAndBGP(projection, dataset, pattern, env);
    } else if(pattern.kind === "UNION") {
        return this.executeUNION(projection, dataset, pattern.value, env);
    } else if(pattern.kind === "JOIN") {
        return this.executeJOIN(projection, dataset, pattern, env);
    } else if(pattern.kind === "LEFT_JOIN") {
        return this.executeLEFT_JOIN(projection, dataset, pattern, env);
    } else if(pattern.kind === "FILTER") {
        // Some components may have the filter inside the unit
        var results = this.executeSelectUnit(projection, dataset, pattern.value, env);
        if(results != null) {
            results = QueryFilters.checkFilters(pattern, results, false, dataset, env, this);
            return results;
        } else {
            return [];
        }
    } else if(pattern.kind === "EMPTY_PATTERN") {
        // as an example of this case  check DAWG test case: algebra/filter-nested-2
        return [];
    } else if(pattern.kind === "ZERO_OR_MORE_PATH" || pattern.kind === 'ONE_OR_MORE_PATH') {
        return this.executeZeroOrMorePath(pattern, dataset, env);
    } else {
        //console.log("Cannot execute query pattern " + pattern.kind + ". Not implemented yet.");
        return null;
    }
};
QueryEngine.QueryEngine.prototype.executeZeroOrMorePath = function(pattern, dataset, env) {
    //console.log("EXECUTING ZERO OR MORE PATH");
    //console.log("X");
    //console.log(pattern.x);
    //console.log("Y");
    //console.log(pattern.y);
    var projection = [],
```

```
        bindings,
        acum,
        results,
        vx,
        intermediate,
        nextBinding,
        vxDenorm,
        origVXName,
        nextPath,
        i,
        j,
        initial,
        pending,
        collected,
        last = null,
        nextOid,
        path,
        value;
    if(pattern.x.token === 'var') {
        projection.push({token: 'variable',
            kind: 'var',
            value: pattern.x.value});
    }
    if(pattern.y.token === 'var') {
        projection.push({token: 'variable',
            kind: 'var',
            value: pattern.y.value});
    }

    if(projection.length === 0) {
        projection.push({"token": "variable", "kind": "*"});
    }
    //console.log("COMPUTED PROJECTION");
    //console.log(projection);
    if(pattern.x.token === 'var' && pattern.y.token === 'var') {
        bindings = this.executeAndBGP(projection, dataset, pattern.path, env);
        //console.log("BINDINGS "+bindings.length);
        //console.log(bindings);
        acum = {};
        results = [];
        origVXName = pattern.x.value;
        last = pattern.x;
        nextPath = pattern.path;
        //console.log("VAR - VAR PATTERN");
        //console.log(nextPath.value);
        for(i=0; i<bindings.length; i++) {
            vx = bindings[i][origVXName];
            if(acum[vx] == null) {
                vxDenorm = this.lexicon.retrieve(vx);
                pattern.x = vxDenorm;
                //console.log("REPLACING");
                //console.log(last);
                //console.log("BY");
                //console.log(vxDenorm);
                //console.log(nextPath.value);
                pattern.path = this.abstractQueryTree.replace(nextPath, last, vxDenorm, env);
                nextPath = TreeUtils.clone(pattern.path);
                intermediate = this.executeZeroOrMorePath(pattern, dataset, env);
                for(j=0; j<intermediate.length; j++) {
                    nextBinding = intermediate[j];
                    nextBinding[origVXName] = vx;
                    results.push(nextBinding)
                }
                last = vxDenorm;
            }
        }
        //console.log("RETURNING VAR - VAR");
```

```
            return results;
        } else if(pattern.x.token !== 'var' && pattern.y.token === 'var') {
            acum = {};
            initial = true;
            pending = [];
            collected = [];
            while (initial == true || pending.length !== 0) {
                //console.log("-- Iteration");
                //console.log(pattern.path.value[0]);
                if(initial === true) {
                    bindings = this.executeAndBGP(projection, dataset, pattern.path, env);
                    //console.log("SAVING LAST");
                    //console.log(pattern.x);
                    last = pattern.x;
                    initial = false;
                } else {
                    nextOid = pending.pop();
                    //console.log("POPPING:"+nextOid);
                    value = this.lexicon.retrieve(nextOid);
                    path = pattern.path; //Utils.clone(pattern.path);
                    //console.log(path.value[0]);
                    //console.log("REPLACING");
                    //console.log(last);
                    //console.log("BY");
                    //console.log(value);
                    path = this.abstractQueryTree.replace(path, last, value, env);
                    //console.log(path.value[0]);
                    bindings = this.executeAndBGP(projection, dataset, path, env);
                    last = value;
                }
                //console.log("BINDINGS!");
                //console.log(bindings);
                for(i=0; i<bindings.length; i++) {
                    //console.log(bindings[i][pattern.y.value])
                    value = bindings[i][pattern.y.value];
                    //console.log("VALUE:"+value);
                    if (acum[value] !== true) {
                        nextBinding = {};
                        nextBinding[pattern.y.value] = value;
                        collected.push(nextBinding);
                        acum[value] = true;
                        pending.push(value);
                    }
                }
            }
            //console.log("RETURNING TERM - VAR");
            //console.log(collected);
            return collected;
        } else {
            throw "Kind of path not supported!";
        }
    };
    QueryEngine.QueryEngine.prototype.executeUNION = function(projection, dataset, patterns, env) {
        var setQuery1 = patterns[0],
            setQuery2 = patterns[1],
            set1,
            set2,
            that = this,
            result;
        if(patterns.length != 2) {
            throw new Error("SPARQL algebra UNION with more than two components");
        }
        set1 = that.executeSelectUnit(projection, dataset, setQuery1, env);
        if(set1==null) {
            return null;
        }
        set2 = that.executeSelectUnit(projection, dataset, setQuery2, env);
```

```
        if(set2==null) {
            return null;
        }
        result = QueryPlan.unionBindings(set1, set2);
        result = QueryFilters.checkFilters(patterns, result, false, dataset, env, that);
        return result;
    };
    QueryEngine.QueryEngine.prototype.executeAndBGP = function(projection, dataset, patterns, env) {
        var that = this;
        var result = QueryPlan.executeAndBGPsDPSize(patterns.value, dataset, this, env);
        if(result!=null) {
            return QueryFilters.checkFilters(patterns, result, false, dataset, env, that);
        } else {
            return null;
        }
    };
    QueryEngine.QueryEngine.prototype.executeLEFT_JOIN = function(projection, dataset, patterns, env) {
        var setQuery1 = patterns.lvalue,
            setQuery2 = patterns.rvalue,
            set1,
            set2,
            that = this,
            acum,
            duplicates,
            idx,
            idxColl,
            i,
            j,
            p;
        //console.log("SET QUERY 1");
        //console.log(setQuery1.value);
        set1 = that.executeSelectUnit(projection, dataset, setQuery1, env);
        if(set1==null) {
            return null;
        }
        //console.log("SET QUERY 2");
        //console.log(setQuery2);
        set2 = that.executeSelectUnit(projection, dataset, setQuery2, env);
        if(set2==null) {
            return null;
        }
        //console.log("\nLEFT JOIN SETS:")
        //console.log(set1)
        //console.log(set2)
        var result = QueryPlan.leftOuterJoinBindings(set1, set2);
        //console.log("---")
        //console.log(result);
        var bindings = QueryFilters.checkFilters(patterns, result, true, dataset, env, that);
        //console.log("---")
        //console.log(bindings)
        //console.log("\r\n")
        if(set1.length>1 && set2.length>1) {
            var vars = [];
            var vars1 = {};
            for(p in set1[0]) {
                if (set1[0].hasOwnProperty(p)) {
                    vars1[p] = true;
                }
            }
            for(p in set2[0]) {
                if(set2[0].hasOwnProperty(p) && vars1[p] != true) {
                    vars.push(p);
                }
            }
            acum = [];
            duplicates = {};
            for(i=0; i<bindings.length; i++) {
```

```
                if(bindings[i]["__nullify__"] === true) {
                    for(j=0; j<vars.length; j++) {
                        bindings[i]["bindings"][vars[j]] = null;
                    }
                    idx = [];
                    idxColl = [];
                    for(p in bindings[i]["bindings"]) {
                        if(bindings[i]["bindings"].hasOwnProperty(p) && bindings[i]["bindings"][p] !=
null) {
                            idx.push(p+bindings[i]["bindings"][p]);
                            idx.sort();
                            idxColl.push(idx.join(""));
                        }
                    }
                    // reject duplicates -> (set union)
                    if(duplicates[idx.join("")]==null) {
                        for(j=0; j<idxColl.length; j++) {
                            //console.log(" - "+idxColl[j])
                            duplicates[idxColl[j]] = true;
                        }
                        ////duplicates[idx.join("")]= true
                        acum.push(bindings[i]["bindings"]);
                    }
                } else {
                    acum.push(bindings[i]);
                    idx = [];
                    idxColl = [];
                    for(p in bindings[i]) {
                        if (bindings[i].hasOwnProperty(p)) {
                            idx.push(p+bindings[i][p]);
                            idx.sort();
                            //console.log(idx.join("") + " -> ok");
                            duplicates[idx.join("")] = true;
                        }
                    }

                }
            }
            return acum;
        } else {
            return bindings;
        }
    };
    QueryEngine.QueryEngine.prototype.executeJOIN = function(projection, dataset, patterns, env) {
        var setQuery1 = patterns.lvalue,
            setQuery2 = patterns.rvalue,
            set1,
            set2,
            that = this,
            p;
        set1 = that.executeSelectUnit(projection, dataset, setQuery1, env);
        if(set1 == null) {
            return null;
        }
        set2 = that.executeSelectUnit(projection, dataset, setQuery2, env);
        if(set2 == null) {
            return null;
        }
        var result = null;
        if(set1.length ===0 || set2.length===0) {
            result = [];
        } else {
            var commonVarsTmp = {};
            var commonVars = [];
            for(p in set1[0]) {
                if (set1[0].hasOwnProperty(p)) {
                    commonVarsTmp[p] = false;
```

```
                }
            }
            for(p in set2[0]) {
                if(set2[0].hasOwnProperty(p) && commonVarsTmp[p] === false) {
                    commonVars.push(p);
                }
            }
            if(commonVars.length == 0) {
                result = QueryPlan.joinBindings(set1, set2);
            } else if(this.abstractQueryTree.treeWithUnion(setQuery1) ||
                this.abstractQueryTree.treeWithUnion(setQuery2)) {
                result = QueryPlan.joinBindings(set1, set2);
            } else {
                result = QueryPlan.joinBindings2(commonVars, set1, set2);
            }
        }
        result = QueryFilters.checkFilters(patterns, result, false, dataset, env, that);
        return result;
    };
    QueryEngine.QueryEngine.prototype.rangeQuery = function(quad, queryEnv) {
        var that = this;
        //console.log("BEFORE:");
        //console.log("QUAD:");
        //console.log(quad);
        var key = that.normalizeQuad(quad, queryEnv, false);
        if(key != null) {
            //console.log("RANGE QUERY:")
            //console.log(key);
            //console.log(new QuadIndexCommon.Pattern(key));
            var quads = that.backend.range(new QuadIndexCommon.Pattern(key));
            //console.log("retrieved");
            //console.log(quads)
            if(quads == null || quads.length == 0) {
                return [];
            } else {
                return quads;
            }
        } else {
            //console.log("ERROR normalizing quad");
            return null;
        }
    };
    // Update queries
    QueryEngine.QueryEngine.prototype.executeUpdate = function(syntaxTree, callback) {
        var prologue = syntaxTree.prologue,
            units = syntaxTree.units,
            that = this,
            i,
            j,
            queryEnv = {
                blanks: {},
                outCache: {}
            },
            quad,
            result;
        this.registerNsInEnvironment(prologue, queryEnv);
        for(i=0; i<units.length; i++) {
            var aqt = that.abstractQueryTree.parseExecutableUnit(units[i]);
            if(aqt.kind === 'insertdata') {
                for(j=0; j<aqt.quads.length; j++) {
                    quad = aqt.quads[j];
                    result = that._executeQuadInsert(quad, queryEnv);
                    if(result !== true) {
                        return callback(false, error);
                    }
                }
                callback(true);
```

```
        } else if(aqt.kind === 'deletedata') {
            for(j=0; j<aqt.quads.length; j++) {
                quad = aqt.quads[j];
                this._executeQuadDelete(quad, queryEnv);
            }
            callback(true);
        } else if(aqt.kind === 'modify') {
            this._executeModifyQuery(aqt, queryEnv, callback);
        } else if(aqt.kind === 'create') {
            callback(true);
        } else if(aqt.kind === 'load') {
            var graph = {
                'uri': TreeUtils.lexicalFormBaseUri(aqt.sourceGraph, queryEnv)
            };
            if(aqt.destinyGraph != null) {
                graph = {
                    'uri': TreeUtils.lexicalFormBaseUri(aqt.destinyGraph, queryEnv)
                };
            }
            //console.log("IN ENGINE, LOAD", aqt.sourceGraph, graph);
            this.rdfLoader.load(aqt.sourceGraph.value, graph, function(success, result){
                //console.log("IN ENGINE, LOAD SUCCESS", success, result);
                if(success == false) {
                    //console.log("Error loading graph");
                    //console.log(result);
                    callback(false, "error batch loading quads");
                } else {
                    that.batchLoad(result.toQuads());
                    callback(result!=null, result||"error batch loading quads");
                }
            });
        } else if(aqt.kind === 'drop') {
            this._executeClearGraph(aqt.destinyGraph, queryEnv, callback);
        } else if(aqt.kind === 'clear') {
            this._executeClearGraph(aqt.destinyGraph, queryEnv, callback);
        } else {
            throw new Error("not supported execution unit");
        }
    }
};
QueryEngine.QueryEngine.prototype.batchLoad = function(quads, callback) {
    var subject = null,
        predicate = null,
        object = null,
        graph = null,
        counter = 0,
        success = true,
        blanks = {},
        maybeBlankOid,
        oid,
        quad,
        key,
        originalQuad,
        i;
    if(this.eventsOnBatchLoad) {
        this.callbacksBackend.startGraphModification();
    }
    for(i=0; i<quads.length; i++) {
        quad = quads[i];
        // subject
        if(quad.subject['uri'] || quad.subject.token === 'uri') {
            oid = this.lexicon.registerUri(quad.subject.uri || quad.subject.value);
            if(quad.subject.uri != null) {
                quad.subject = {'token': 'uri', 'value': quad.subject.uri};
                delete quad.subject['uri'];
            }
            subject = oid;
```

```
            } else if(quad.subject['literal'] || quad.subject.token === 'literal') {
                oid = this.lexicon.registerLiteral(quad.subject.literal || quad.subject.value);
                if(quad.subject.literal != null) {
                    quad.subject = this.lexicon.parseLiteral(quad.subject.literal);
                    delete quad.subject['literal'];
                }
                subject = oid;
            } else {
                maybeBlankOid = blanks[quad.subject.blank || quad.subject.value];
                if(maybeBlankOid == null) {
                    maybeBlankOid = this.lexicon.registerBlank(quad.subject.blank || quad.subject.value);
                    blanks[(quad.subject.blank || quad.subject.value)] = maybeBlankOid;
                }
                if(quad.subject.token == null) {
                    quad.subject.token = 'blank';
                    quad.subject.value = quad.subject.blank;
                    delete quad.subject['blank'];
                }
                subject = maybeBlankOid;
            }
            // predicate
            if(quad.predicate['uri'] || quad.predicate.token === 'uri') {
                oid = this.lexicon.registerUri(quad.predicate.uri || quad.predicate.value);
                if(quad.predicate.uri != null) {
                    quad.predicate = {'token': 'uri', 'value': quad.predicate.uri};
                    delete quad.subject['uri'];
                }
                predicate = oid;
            } else if(quad.predicate['literal'] || quad.predicate.token === 'literal') {
                oid = this.lexicon.registerLiteral(quad.predicate.literal || quad.predicate.value);
                if(quad.predicate.literal != null) {
                    quad.predicate = this.lexicon.parseLiteral(quad.predicate.literal);
                    delete quad.predicate['literal'];
                }
                predicate = oid;
            } else {
                maybeBlankOid = blanks[quad.predicate.blank || quad.predicate.value];
                if(maybeBlankOid == null) {
                    maybeBlankOid = this.lexicon.registerBlank(quad.predicate.blank ||
quad.predicate.value);
                    blanks[(quad.predicate.blank || quad.predicate.value)] = maybeBlankOid;
                }
                if(quad.predicate.token == null) {
                    quad.predicate.token = 'blank';
                    quad.predicate.value = quad.predicate.blank;
                    delete quad.predicate['blank'];
                }
                predicate = maybeBlankOid;
            }
            // object
            if(quad.object['uri'] || quad.object.token === 'uri') {
                oid = this.lexicon.registerUri(quad.object.uri || quad.object.value);
                if(quad.object.uri != null) {
                    quad.object = {'token': 'uri', 'value': quad.object.uri};
                    delete quad.subject['uri'];
                }
                object = oid;
            } else if(quad.object['literal'] || quad.object.token === 'literal') {
                if(quad.object.token === 'literal') {
                    if(quad.object.type != null) {
                        quad.object.value = '"'+quad.object.value+'"^^<'+quad.object.type+'>';
                    } else if(quad.object.lang != null) {
                        quad.object.value = '"'+quad.object.value+'"@'+quad.object.lang;
                    } else {
                        quad.object.value = '"'+quad.object.value+'"';
                    }
                }
```

```
        oid = this.lexicon.registerLiteral(quad.object.literal || quad.object.value);
        if(quad.object.literal != null) {
            quad.object = this.lexicon.parseLiteral(quad.object.literal);
            delete quad.object['literal'];
        }
        object = oid;
    } else {
        maybeBlankOid = blanks[quad.object.blank || quad.object.value];
        if(maybeBlankOid == null) {
            maybeBlankOid = this.lexicon.registerBlank(quad.object.blank || quad.object.value);
            blanks[(quad.object.blank || quad.object.value)] = maybeBlankOid;
        }
        if(quad.object.token == null) {
            quad.object.token = 'blank';
            quad.object.value = quad.object.blank;
            delete quad.object['blank'];
        }
        object = maybeBlankOid;
    }
    // graph
    if(quad.graph['uri'] || quad.graph.token === 'uri') {
        oid = this.lexicon.registerUri(quad.graph.uri || quad.graph.value);
        if(quad.graph.uri != null) {
            quad.graph = {'token': 'uri', 'value': quad.graph.uri};
            delete quad.subject['uri'];
        }
        this.lexicon.registerGraph(oid);
        graph = oid;
    } else if(quad.graph['literal'] || quad.graph.token === 'literal') {
        oid = this.lexicon.registerLiteral(quad.graph.literal || quad.graph.value);
        if(quad.predicate.literal != null) {
            quad.predicate = this.lexicon.parseLiteral(quad.predicate.literal);
            delete quad.predicate['literal'];
        }
        graph = oid;
    } else {
        maybeBlankOid = blanks[quad.graph.blank || quad.graph.value];
        if(maybeBlankOid == null) {
            maybeBlankOid = this.lexicon.registerBlank(quad.graph.blank || quad.graph.value);
            blanks[(quad.graph.blank || quad.graph.value)] = maybeBlankOid;
        }
        if(quad.graph.token == null) {
            quad.graph.token = 'blank';
            quad.graph.value = quad.graph.blank;
            delete quad.graph['blank'];
        }
        graph = maybeBlankOid;
    }
    originalQuad = quad;
    quad = {
        subject: subject,
        predicate: predicate,
        object: object,
        graph: graph
    };
    key = new QuadIndexCommon.NodeKey(quad);
    var result = this.backend.search(key);
    if(!result) {
        result = this.backend.index(key);
        if(result == true){
            if(this.eventsOnBatchLoad)
                this.callbacksBackend.nextGraphModification(Callbacks.added, [originalQuad,quad]);
            counter = counter + 1;
        } else {
            success = false;
            break;
        }
```

```
            }
        }
        if(this.lexicon["updateAfterWrite"] != null) {
            this.lexicon["updateAfterWrite"]();
        }
        var exitFn = function(){
            if(success) {
                if(callback)
                    callback(true, counter);
            } else {
                if(callback)
                    callback(false, null);
            }
        };
        if(this.eventsOnBatchLoad) {
            this.callbacksBackend.endGraphModification(function(){
                exitFn();
            });
        } else {
            exitFn();
        }
        if(success) {
            return counter;
        } else {
            return null;
        }
    };
    // @modified dp
    QueryEngine.QueryEngine.prototype.computeCosts = function (quads, env) {
        for (var i = 0; i < quads.length; i++) {
            quads[i]['_cost'] = this.quadCost(quads[i], env);
        }

        return quads;
    };
    // Low level operations for update queries
    QueryEngine.QueryEngine.prototype._executeModifyQuery = function(aqt, queryEnv, callback) {
        var that = this,
            querySuccess = true,
            bindings = null,
            components = ['subject', 'predicate', 'object', 'graph'],
            component,
            c,
            i,
            j,
            binding,
            quads,
            quad;
        aqt.insert = aqt.insert == null ? [] : aqt.insert;
        aqt.delete = aqt.delete == null ? [] : aqt.delete;
        TreeUtils.seq(function(k) {
                // select query
                var defaultGraph = [];
                var namedGraph = [];
                if(aqt.with != null) {
                    defaultGraph.push(aqt.with);
                }
                if(aqt.using != null) {
                    namedGraph = [];
                    for(var i=0; i<aqt.using.length; i++) {
                        var usingGraph = aqt.using[i];
                        if(usingGraph.kind === 'named') {
                            namedGraph.push(usingGraph.uri);
                        } else {
                            defaultGraph.push(usingGraph.uri);
                        }
                    }
```

```
            }
            aqt.dataset = {};
            aqt.projection = [{"token": "variable", "kind": "*"}];
            that.executeSelect(aqt, queryEnv, defaultGraph, namedGraph, function(success, result) {
                if(success) {
                    result = that.denormalizeBindingsList(result, queryEnv);
                    if(result!=null) {
                        bindings = result;
                    } else {
                        querySuccess = false;
                    }
                    return k();
                } else {
                    querySuccess = false;
                    return k();
                }
            });
        }, function(k) {
            // delete query
            var defaultGraph = aqt.with;
            if(querySuccess) {
                var quads = [];
                for(i=0; i<aqt.delete.length; i++) {
                    var src = aqt.delete[i];
                    for(j=0; j<bindings.length; j++) {
                        quad = {};
                        binding = bindings[j];
                        for(var c=0; c<components.length; c++) {
                            var component = components[c];
                            if(component == 'graph' && src[component] == null) {
                                quad['graph'] = defaultGraph;
                            } else if(src[component].token === 'var') {
                                quad[component] = binding[src[component].value];
                            } else {
                                quad[component] = src[component];
                            }
                        }
                        quads.push(quad);
                    }
                }
                var quad;
                for(j=0; j<quads.length; j++) {
                    quad = quads[j];
                    that._executeQuadDelete(quad, queryEnv);
                }
                k();
            } else {
                k();
            }
        },function(k) {
            // insert query
            var defaultGraph = aqt.with;
            if(querySuccess) {
                quads = [];
                for(i=0; i<aqt.insert.length; i++) {
                    var src = aqt.insert[i];
                    for(j=0; j<bindings.length; j++) {
                        quad = {};
                        binding = bindings[j];
                        for(c=0; c<components.length; c++) {
                            component = components[c];
                            if(component == 'graph' && src[component] == null) {
                                quad['graph'] = defaultGraph;
                            } else if(src[component].token === 'var') {
                                quad[component] = binding[src[component].value];
                            } else {
                                quad[component] = src[component];
```

```
                            }
                        }
                        quads.push(quad);
                    }
                }
                for(i=0; i<quads.length; i++) {
                    quad = quads[i];
                    that._executeQuadInsert(quad, queryEnv);
                }
                k();
            } else {
                k();
            }
        }
    )(function(){
        callback(querySuccess);
    });
};
QueryEngine.QueryEngine.prototype._executeQuadInsert = function(quad, queryEnv) {
    var that = this,
        normalized = this.normalizeQuad(quad, queryEnv, true),
        result,
        key;
    if(normalized != null) {
        key = new QuadIndexCommon.NodeKey(normalized);
        result = that.backend.search(key);
        if(result){
            return(result);
        } else {
            result = that.backend.index(key);
            if(result == true){
                that.callbacksBackend.nextGraphModification(Callbacks.added, [quad, normalized]);
                return true;
            } else {
                //console.log("ERROR inserting quad");
                return false;
            }
        }
    } else {
        //console.log("ERROR normalizing quad");
        return false;
    }
};
QueryEngine.QueryEngine.prototype._executeQuadDelete = function(quad, queryEnv) {
    var that = this;
    var normalized = this.normalizeQuad(quad, queryEnv, false);
    if(normalized != null) {
        var key = new QuadIndexCommon.NodeKey(normalized);
        that.backend.delete(key);
        var result = that.lexicon.unregister(quad, key);
        if(result == true){
            that.callbacksBackend.nextGraphModification(Callbacks['deleted'], [quad, normalized]);
            return true;
        } else {
            //console.log("ERROR unregistering quad");
            return false;
        }
    } else {
        //console.log("ERROR normalizing quad");
        return false;
    }
};
QueryEngine.QueryEngine.prototype._executeClearGraph = function(destinyGraph, queryEnv, callback) {
    var that = this,
        graphs,
        foundErrorDeleting,
        graph,
```

```
            floop,
            graphUri;
        if(destinyGraph === 'default') {
            this.execute("DELETE { ?s ?p ?o } WHERE { ?s ?p ?o }", callback);
        } else if(destinyGraph === 'named') {
            graphs = this.lexicon.registeredGraphs(true);
            if(graphs!=null) {
                foundErrorDeleting = false;
                TreeUtils.repeat(0, graphs.length,function(k,env) {
                    graph = graphs[env._i];
                    floop = arguments.callee;
                    if(!foundErrorDeleting) {
                        that.execute("DELETE { GRAPH <"+graph+"> { ?s ?p ?o } } WHERE { GRAPH <"+graph+">
{ ?s ?p ?o } }", function(success){
                            foundErrorDeleting = !success;
                            k(floop, env);
                        });
                    } else {
                        k(floop, env);
                    }
                }, function() {
                    callback(!foundErrorDeleting);
                });
            } else {
                callback(false, "Error deleting named graphs");
            }
        } else if(destinyGraph === 'all') {
            this.execute("CLEAR DEFAULT", function(success, result) {
                if(success) {
                    that.execute("CLEAR NAMED", callback);
                } else {
                    callback(false,result);
                }
            });
        } else {
            // destinyGraph is an URI
            if(destinyGraph.token == 'uri') {
                graphUri = TreeUtils.lexicalFormBaseUri(destinyGraph,queryEnv);
                if(graphUri != null) {
                    this.execute("DELETE { GRAPH <"+graphUri+"> { ?s ?p ?o } } WHERE { GRAPH <"+graphUri
+"> { ?s ?p ?o } }", callback);
                } else {
                    callback(false, "wrong graph URI");
                }
            } else {
                callback(false, "wrong graph URI");
            }
        }
    };
    QueryEngine.QueryEngine.prototype.checkGroupSemantics = function(groupVars, projectionVars) {
        if(groupVars === 'singleGroup') {
            return true;
        }
        var projection = {};
        for(var i=0; i<groupVars.length; i++) {
            var groupVar = groupVars[i];
            if(groupVar.token === 'var') {
                projection[groupVar.value] = true;
            } else if(groupVar.token === 'aliased_expression') {
                projection[groupVar.alias.value] = true;
            }
        }
        for(i=0; i<projectionVars.length; i++) {
            var projectionVar = projectionVars[i];
            if(projectionVar.kind === 'var') {
                if(projection[projectionVar.value.value] == null) {
                    return false;
```

```
                }
            } else if(projectionVar.kind === 'aliased' &&
                projectionVar.expression &&
                projectionVar.expression.primaryexpression === 'var') {
                if(projection[projectionVar.expression.value.value] == null) {
                    return false;
                }
            }
        }
        return true;
    };
    return QueryEngine;
});

define([
    "./../../graphite/utils",
    "./../trees/utils"
], function (Utils, TreeUtils) {
    var QueryFilters = {};
    QueryFilters.checkFilters = function(pattern, bindings, nullifyErrors, dataset, queryEnv,
queryEngine) {
        var filters = pattern.filter;
        var nullified = [];
        if(filters==null || pattern.length != null) {
            return bindings;
        }
        for(var i=0; i<filters.length; i++) {
            var filter = filters[i];
            var filteredBindings = QueryFilters.run(filter.value, bindings, nullifyErrors, dataset,
queryEnv, queryEngine);
            var acum = [];
            for(var j=0; j<filteredBindings.length; j++) {
                if(filteredBindings[j]["__nullify__"]!=null) {
                    nullified.push(filteredBindings[j]);
                } else {
                    acum.push(filteredBindings[j]);
                }
            }
            bindings = acum;
        }
        return bindings.concat(nullified);
    };
    QueryFilters.boundVars = function(filterExpr) {
        var acum;
        if(filterExpr.expressionType != null) {
            var expressionType = filterExpr.expressionType;
            if(expressionType == 'relationalexpression') {
                var op1 = filterExpr.op1;
                var op2 = filterExpr.op2;
                return QueryFilters.boundVars(op1)+QueryFilters.boundVars(op2);
            } else if(expressionType == 'conditionalor' || expressionType == 'conditionaland') {
                return Utils.map(filterExpr.operands, function (operand) {
                    return QueryFilters.boundVars(operand);
                });
            } else if(expressionType == 'builtincall') {
                if(filterExpr.args == null) {
                    return [];
                }
                return Utils.map(filterExpr.args, function (arg) {
                    return QueryFilters.boundVars(arg);
                });
            } else if(expressionType == 'multiplicativeexpression') {
                acum = QueryFilters.boundVars(filterExpr.factor);
                Utils.each(filterExpr.factors, function (factor) {
                    acum = acum.concat(QueryFilters.boundVars(factor.expression))
                });
                return acum;
```

```
        } else if(expressionType == 'additiveexpression') {
            acum = QueryFilters.boundVars(filterExpr.summand);
            Utils.each(filterExpr.summands, function (summand) {
                acum = acum.concat(QueryFilters.boundVars(summand.expression));
            });
            return acum;
        } else if(expressionType == 'regex') {
            acum = QueryFilters.boundVars(filterExpr["expression1"]);
            return acum.concat(QueryFilters.boundVars(filterExpr["expression2"]));
        } else if(expressionType == 'unaryexpression') {
            return QueryFilters.boundVars(filterExpr.expression);
        } else if(expressionType == 'atomic') {
            if(filterExpr.primaryexpression == 'var') {
                return [filterExpr.value];
            } else {
                // numeric, literal, etc...
                return [];
            }
        }
    } else {
        //console.log("ERROR");
        //console.log(filterExpr);
        throw("Cannot find bound expressions in a no expression token");
    }
};
QueryFilters.run = function(filterExpr, bindings, nullifyFilters, dataset, env, queryEngine) {
    var denormBindings = queryEngine.copyDenormalizedBindings(bindings, env.outCache);
    var filteredBindings = [];
    var ebv;
    var thisDenormBindings;
    Utils.each(bindings, function (binding, i) {
        thisDenormBindings = denormBindings[i];
        ebv = QueryFilters.runFilter(filterExpr, thisDenormBindings, queryEngine, dataset, env);
        // ebv can be directly a RDFTerm (e.g. atomic expression in filter)
        // this additional call to ebv will return -> true/false/error
        ebv = QueryFilters.ebv(ebv);
        //console.log("EBV:")
        //console.log(ebv)
        //console.log("FOR:")
        //console.log(thisDenormBindings)
        if(QueryFilters.isEbvError(ebv)) {
            // error
            if(nullifyFilters) {
                filteredBindings.push({"__nullify__": true, "bindings": binding});
            }
        } else if(ebv === true) {
            // true
            filteredBindings.push(binding);
        } else {
            // false
            if(nullifyFilters) {
                filteredBindings.push({"__nullify__": true, "bindings": binding});
            }
        }
    });
    return filteredBindings;
};
QueryFilters.collect = function(filterExpr, bindings, dataset, env, queryEngine) {
    var denormBindings = queryEngine.copyDenormalizedBindings(bindings, env.outCache);
    var filteredBindings = [];
    for(var i=0; i<denormBindings.length; i++) {
        var thisDenormBindings = denormBindings[i];
        var ebv = QueryFilters.runFilter(filterExpr, thisDenormBindings, queryEngine, dataset, env);
        filteredBindings.push({binding:bindings[i], value:ebv});
    }
    return(filteredBindings);
};
```

```
    // @todo add more aggregation functions here
    QueryFilters.runAggregator = function(aggregator, bindingsGroup, queryEngine, dataset, env) {
        if(bindingsGroup == null || bindingsGroup.length === 0) {
            return QueryFilters.ebvError();
        } else if(aggregator.token === 'variable' && aggregator.kind == 'var') {
            return bindingsGroup[0][aggregator.value.value];
        } else if(aggregator.token === 'variable' && aggregator.kind === 'aliased') {
            if(aggregator.expression.expressionType === 'atomic' &&
aggregator.expression.primaryexpression === 'var') {
                return bindingsGroup[0][aggregator.expression.value.value];
            } else if(aggregator.expression.expressionType === 'aggregate') {
                if(aggregator.expression.aggregateType === 'max') {
                    var max = null;
                    Utils.each(bindingsGroup, function (bindings) {
                        var ebv = QueryFilters.runFilter(aggregator.expression.expression, bindings,
queryEngine, dataset, env);
                        if(!QueryFilters.isEbvError(ebv)) {
                            if(max === null) {
                                max = ebv;
                            } else {
                                if(QueryFilters.runLtFunction(max, ebv).value === true) {
                                    max = ebv;
                                }
                            }
                        }
                    });
                    if(max===null) {
                        return QueryFilters.ebvError();
                    } else {
                        return max;
                    }
                } else if(aggregator.expression.aggregateType === 'min') {
                    var min = null;
                    Utils.each(bindingsGroup, function (bindings) {
                        var ebv = QueryFilters.runFilter(aggregator.expression.expression, bindings,
queryEngine, dataset, env);
                        if(!QueryFilters.isEbvError(ebv)) {
                            if(min === null) {
                                min = ebv;
                            } else {
                                if(QueryFilters.runGtFunction(min, ebv).value === true) {
                                    min = ebv;
                                }
                            }
                        }
                    });
                    if(min===null) {
                        return QueryFilters.ebvError();
                    } else {
                        return min;
                    }
                } else if(aggregator.expression.aggregateType === 'count') {
                    var distinct = {};
                    var count = 0;
                    if(aggregator.expression.expression === '*') {
                        if(aggregator.expression.distinct != null && aggregator.expression.distinct !=
'') {
                            Utils.each(bindingsGroup, function (bindings) {
                                var key = TreeUtils.hashTerm(bindings);
                                if(distinct[key] == null) {
                                    distinct[key] = true;
                                    count++;
                                }
                            });
                        } else {
                            count = bindingsGroup.length;
                        }
```

```
                    } else {
                        Utils.each(bindingsGroup, function (bindings) {
                            var ebv = QueryFilters.runFilter(aggregator.expression.expression, bindings,
queryEngine, dataset, env);
                            if(!QueryFilters.isEbvError(ebv)) {
                                if(aggregator.expression.distinct != null &&
aggregator.expression.distinct != '') {
                                    var key = TreeUtils.hashTerm(ebv);
                                    if(distinct[key] == null) {
                                        distinct[key] = true;
                                        count++;
                                    }
                                } else {
                                    count++;
                                }
                            }
                        });
                    }
                    return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#integer",
value:''+count};
                } else if(aggregator.expression.aggregateType === 'avg') {
                    var distinct = {};
                    var aggregated = {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#integer",
value:'0'};
                    var count = 0;
                    for(var i=0; i< bindingsGroup.length; i++) {
                        var bindings = bindingsGroup[i];
                        var ebv = QueryFilters.runFilter(aggregator.expression.expression, bindings,
queryEngine, dataset, env);
                        if(!QueryFilters.isEbvError(ebv)) {
                            if(aggregator.expression.distinct != null && aggregator.expression.distinct !
= '') {
                                var key = TreeUtils.hashTerm(ebv);
                                if(distinct[key] == null) {
                                    distinct[key] = true;
                                    if(QueryFilters.isNumeric(ebv)) {
                                        aggregated = QueryFilters.runSumFunction(aggregated, ebv);
                                        count++;
                                    }
                                }
                            } else {
                                if(QueryFilters.isNumeric(ebv)) {
                                    aggregated = QueryFilters.runSumFunction(aggregated, ebv);
                                    count++;
                                }
                            }
                        }
                    }

                    var result = QueryFilters.runDivFunction(aggregated, {token: 'literal', type:"http://
www.w3.org/2001/XMLSchema#integer", value:''+count});
                    result.value = ''+result.value;
                    return result;
                } else if(aggregator.expression.aggregateType === 'sum') {
                    //console.log("IN QUERY FILTERS");
                    var distinct = {};
                    var aggregated = {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#integer",
value:'0'};
                    for(var i=0; i< bindingsGroup.length; i++) {
                        var bindings = bindingsGroup[i];
                        var ebv = QueryFilters.runFilter(aggregator.expression.expression, bindings,
queryEngine, dataset, env);
                        if(!QueryFilters.isEbvError(ebv)) {
                            if(aggregator.expression.distinct != null && aggregator.expression.distinct !
= '') {
                                var key = TreeUtils.hashTerm(ebv);
                                if(distinct[key] == null) {
```

```
                                    distinct[key] = true;
                                    if(QueryFilters.isNumeric(ebv)) {
                                        aggregated = QueryFilters.runSumFunction(aggregated, ebv);
                                    }
                                }
                            } else {
                                if(QueryFilters.isNumeric(ebv)) {
                                    aggregated = QueryFilters.runSumFunction(aggregated, ebv);
                                }
                            }
                        }
                    }
                    aggregated.value =''+aggregated.value;
                    return aggregated;
                } else {
                    var ebv = QueryFilters.runFilter(aggregate.expression, bindingsGroup[0], dataset,
{blanks:{}, outCache:{}});
                    return ebv;
                }
            }
        }
    };

    QueryFilters.runFilter = function(filterExpr, bindings, queryEngine, dataset, env) {
        if(filterExpr.expressionType != null) {
            var expressionType = filterExpr.expressionType;
            if(expressionType == 'relationalexpression') {
                var op1 = QueryFilters.runFilter(filterExpr.op1, bindings,queryEngine, dataset, env);
                var op2 = QueryFilters.runFilter(filterExpr.op2, bindings,queryEngine, dataset, env);
                return QueryFilters.runRelationalFilter(filterExpr, op1, op2, bindings, queryEngine,
dataset, env);
            } else if(expressionType == 'conditionalor') {
                return QueryFilters.runOrFunction(filterExpr, bindings, queryEngine, dataset, env);
            } else if (expressionType == 'conditionaland') {
                return QueryFilters.runAndFunction(filterExpr, bindings, queryEngine, dataset, env);
            } else if(expressionType == 'additiveexpression') {
                return QueryFilters.runAddition(filterExpr.summand, filterExpr.summands, bindings,
queryEngine, dataset, env);
            } else if(expressionType == 'builtincall') {
                return QueryFilters.runBuiltInCall(filterExpr.builtincall, filterExpr.args, bindings,
queryEngine, dataset, env);
            } else if(expressionType == 'multiplicativeexpression') {
                return QueryFilters.runMultiplication(filterExpr.factor, filterExpr.factors, bindings,
queryEngine, dataset, env);
            } else if(expressionType == 'unaryexpression') {
                return QueryFilters.runUnaryExpression(filterExpr.unaryexpression, filterExpr.expression,
bindings, queryEngine, dataset, env);
            } else if(expressionType == 'irireforfunction') {
                return QueryFilters.runIriRefOrFunction(filterExpr.iriref, filterExpr.args, bindings,
queryEngine, dataset, env);
            } else if(expressionType == 'regex') {
                return QueryFilters.runRegex(filterExpr.text, filterExpr.pattern, filterExpr.flags,
bindings, queryEngine, dataset, env)
            } else if(expressionType == 'atomic') {
                if(filterExpr.primaryexpression == 'var') {
                    // lookup the var in the bindings
                    var val = bindings[filterExpr.value.value];
                    return val;
                } else {
                    // numeric, literal, etc...
                    //return queryEngine.filterExpr.value;
                    if(typeof(filterExpr.value) != 'object') {
                        return filterExpr.value
                    } else {
                        if(filterExpr.value.type == null || typeof(filterExpr.value.type) != 'object') {
                            return filterExpr.value
                        } else {
```

```
                            // type can be parsed as a hash using namespaces

                            filterExpr.value.type =  TreeUtils.lexicalFormBaseUri(filterExpr.value.type,
env);
                            return filterExpr.value
                        }
                    }
                }
            } else {
                throw("Unknown filter expression type");
            }
        } else {
            throw("Cannot find bound expressions in a no expression token");
        }
    };

    QueryFilters.isRDFTerm = function(val) {
        if(val==null) {
            return false;
        } if((val.token && val.token == 'literal') ||
            (val.token && val.token == 'uri') ||
            (val.token && val.token == 'blank')) {
            return true;
        } else {
            return false;
        }
    };


    /*
     17.4.1.7 RDFterm-equal

     xsd:boolean   RDF term term1 = RDF term term2

     Returns TRUE if term1 and term2 are the same RDF term as defined in Resource Description Framework
(RDF):
     Concepts and Abstract Syntax [CONCEPTS]; produces a type error if the arguments are both literal but
are not
     the same RDF term *; returns FALSE otherwise. term1 and term2 are the same if any of the following
is true:

     term1 and term2 are equivalent IRIs as defined in 6.4 RDF URI References of [CONCEPTS].
     term1 and term2 are equivalent literals as defined in 6.5.1 Literal Equality of [CONCEPTS].
     term1 and term2 are the same blank node as described in 6.6 Blank Nodes of [CONCEPTS].
     */
    QueryFilters.RDFTermEquality = function(v1, v2, queryEngine, env) {
        if(v1.token === 'literal' && v2.token === 'literal') {
            if(v1.lang == v2.lang && v1.type == v2.type && v1.value == v2.value) {

                return true;
            } else {


                if(v1.type != null && v2.type != null) {
                    return  QueryFilters.ebvError();
                } else if(QueryFilters.isSimpleLiteral(v1) && v2.type!=null){
                    return QueryFilters.ebvError();
                } else if(QueryFilters.isSimpleLiteral(v2) && v1.type!=null){
                    return QueryFilters.ebvError();
                } else {
                    return false;
                }

//              if(v1.value != v2.value) {
//                  return QueryFilters.ebvError();
//              } else if(v1.type && v2.type && v1.type!=v2.type) {
//                  return QueryFilters.ebvError();
```

```
//            } else if(QueryFilters.isSimpleLiteral(v1) && v2.type!=null){
//                return QueryFilters.ebvError();
//            } else if(QueryFilters.isSimpleLiteral(v2) && v1.type!=null){
//                return QueryFilters.ebvError();
//            } else {
//                return false;
//            }

            }
        } else if(v1.token === 'uri' && v2.token === 'uri') {
            return TreeUtils.lexicalFormBaseUri(v1, env) == TreeUtils.lexicalFormBaseUri(v2, env);
        } else if(v1.token === 'blank' && v2.token === 'blank') {
            return v1.value == v2.value;
        } else {
            return false;
        }
    };


    QueryFilters.isInteger = function(val) {
        if(val == null) {
            return false;
        }
        if(val.token === 'literal') {
            if(val.type == "http://www.w3.org/2001/XMLSchema#integer" ||
                val.type == "http://www.w3.org/2001/XMLSchema#decimal" ||
                val.type == "http://www.w3.org/2001/XMLSchema#double" ||
                val.type == "http://www.w3.org/2001/XMLSchema#nonPositiveInteger" ||
                val.type == "http://www.w3.org/2001/XMLSchema#negativeInteger" ||
                val.type == "http://www.w3.org/2001/XMLSchema#long" ||
                val.type == "http://www.w3.org/2001/XMLSchema#int" ||
                val.type == "http://www.w3.org/2001/XMLSchema#short" ||
                val.type == "http://www.w3.org/2001/XMLSchema#byte" ||
                val.type == "http://www.w3.org/2001/XMLSchema#nonNegativeInteger" ||
                val.type == "http://www.w3.org/2001/XMLSchema#unsignedLong" ||
                val.type == "http://www.w3.org/2001/XMLSchema#unsignedInt" ||
                val.type == "http://www.w3.org/2001/XMLSchema#unsignedShort" ||
                val.type == "http://www.w3.org/2001/XMLSchema#unsignedByte" ||
                val.type == "http://www.w3.org/2001/XMLSchema#positiveInteger" ) {
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    };

    QueryFilters.isFloat = function(val) {
        if(val == null) {
            return false;
        }
        if(val.token === 'literal') {
            if(val.type == "http://www.w3.org/2001/XMLSchema#float") {
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    };

    QueryFilters.isDecimal = function(val) {
        if(val == null) {
            return false;
        }
```

```
    if(val.token === 'literal') {
        if(val.type == "http://www.w3.org/2001/XMLSchema#decimal") {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
};

QueryFilters.isDouble = function(val) {
    if(val == null) {
        return false;
    }
    if(val.token === 'literal') {
        if(val.type == "http://www.w3.org/2001/XMLSchema#double") {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
};


QueryFilters.isNumeric = function(val) {
    if(val == null) {
        return false;
    }
    if(val.token === 'literal') {
        if(val.type == "http://www.w3.org/2001/XMLSchema#integer" ||
            val.type == "http://www.w3.org/2001/XMLSchema#decimal" ||
            val.type == "http://www.w3.org/2001/XMLSchema#float" ||
            val.type == "http://www.w3.org/2001/XMLSchema#double" ||
            val.type == "http://www.w3.org/2001/XMLSchema#nonPositiveInteger" ||
            val.type == "http://www.w3.org/2001/XMLSchema#negativeInteger" ||
            val.type == "http://www.w3.org/2001/XMLSchema#long" ||
            val.type == "http://www.w3.org/2001/XMLSchema#int" ||
            val.type == "http://www.w3.org/2001/XMLSchema#short" ||
            val.type == "http://www.w3.org/2001/XMLSchema#byte" ||
            val.type == "http://www.w3.org/2001/XMLSchema#nonNegativeInteger" ||
            val.type == "http://www.w3.org/2001/XMLSchema#unsignedLong" ||
            val.type == "http://www.w3.org/2001/XMLSchema#unsignedInt" ||
            val.type == "http://www.w3.org/2001/XMLSchema#unsignedShort" ||
            val.type == "http://www.w3.org/2001/XMLSchema#unsignedByte" ||
            val.type == "http://www.w3.org/2001/XMLSchema#positiveInteger" ) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
};

QueryFilters.isSimpleLiteral = function(val) {
    if(val && val.token == 'literal') {
        if(val.type == null && val.lang == null) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
```

```
};

QueryFilters.isXsdType = function(type, val) {
    if(val && val.token == 'literal') {
        return val.type == "http://www.w3.org/2001/XMLSchema#"+type;
    } else {
        return false;
    }
};

QueryFilters.ebv = function (term) {
    if (term == null || QueryFilters.isEbvError(term)) {
        return QueryFilters.ebvError();
    } else {
        if (term.token && term.token === 'literal') {
            if (term.type == "http://www.w3.org/2001/XMLSchema#integer" ||
                term.type == "http://www.w3.org/2001/XMLSchema#decimal" ||
                term.type == "http://www.w3.org/2001/XMLSchema#double" ||
                term.type == "http://www.w3.org/2001/XMLSchema#nonPositiveInteger" ||
                term.type == "http://www.w3.org/2001/XMLSchema#negativeInteger" ||
                term.type == "http://www.w3.org/2001/XMLSchema#long" ||
                term.type == "http://www.w3.org/2001/XMLSchema#int" ||
                term.type == "http://www.w3.org/2001/XMLSchema#short" ||
                term.type == "http://www.w3.org/2001/XMLSchema#byte" ||
                term.type == "http://www.w3.org/2001/XMLSchema#nonNegativeInteger" ||
                term.type == "http://www.w3.org/2001/XMLSchema#unsignedLong" ||
                term.type == "http://www.w3.org/2001/XMLSchema#unsignedInt" ||
                term.type == "http://www.w3.org/2001/XMLSchema#unsignedShort" ||
                term.type == "http://www.w3.org/2001/XMLSchema#unsignedByte" ||
                term.type == "http://www.w3.org/2001/XMLSchema#positiveInteger") {
                var tmp = parseFloat(term.value);
                if (isNaN(tmp)) {
                    return false;
                } else {
                    return parseFloat(term.value) != 0;
                }
            } else if (term.type === "http://www.w3.org/2001/XMLSchema#boolean") {
                return (term.value === 'true' || term.value === true || term.value === 'True');
            } else if (term.type === "http://www.w3.org/2001/XMLSchema#string") {
                return term.value != "";
            } else if (term.type === "http://www.w3.org/2001/XMLSchema#dateTime") {
                return (new Date(term.value)) != null;
            } else if (QueryFilters.isEbvError(term)) {
                return term;
            } else if (term.type == null) {
                if (term.value != "") {
                    return true;
                } else {
                    return false;
                }
            } else {
                return QueryFilters.ebvError();
            }
        } else {
            return term.value === true;
        }
    }
};


QueryFilters.ebvTrue = function() {
    var val = {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#boolean", value:true};
    return val;
};

QueryFilters.ebvFalse = function() {
    var val = {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#boolean", value:false};
```

```
        return val;
    };

    QueryFilters.ebvError = function() {
        var val = {token: 'literal', type:"https://github.com/antoniogarrote/js-tools/types#error",
value:null};
        return val;
    };

    QueryFilters.isEbvError = function(term) {
        if(typeof(term) == 'object' && term != null) {
            return term.type === "https://github.com/antoniogarrote/js-tools/types#error";
//    } else if(term == null) {
//        return true;
        } else {
            return false;
        }
    };

    QueryFilters.ebvBoolean = function (bool) {
        if (QueryFilters.isEbvError(bool)) {
            return bool;
        } else {
            if (bool === true) {
                return QueryFilters.ebvTrue();
            } else {
                return QueryFilters.ebvFalse();
            }
        }
    };


    QueryFilters.runRelationalFilter = function(filterExpr, op1, op2, bindings, queryEngine, dataset,
env) {
        var operator = filterExpr.operator;
        if(operator === '=') {
            return QueryFilters.runEqualityFunction(op1, op2, bindings, queryEngine, dataset, env);
        } else if(operator === '!=') {
            var res = QueryFilters.runEqualityFunction(op1, op2, bindings, queryEngine, dataset, env);
            if(QueryFilters.isEbvError(res)) {
                return res;
            } else {
                res.value = !res.value;
                return res;
            }
        } else if(operator === '<') {
            return QueryFilters.runLtFunction(op1, op2, bindings);
        } else if(operator === '>') {
            return QueryFilters.runGtFunction(op1, op2, bindings);
        } else if(operator === '<=') {
            return QueryFilters.runLtEqFunction(op1, op2, bindings);
        } else if(operator === '>=') {
            return QueryFilters.runGtEqFunction(op1, op2, bindings);
        } else {
            throw("Error applying relational filter, unknown operator");
        }
    };

    /**
     * Transforms a JS object representing a [typed] literal in a javascript
     * value that can be used in javascript operations and functions
     */
    QueryFilters.effectiveTypeValue = function(val){
        if(val.token == 'literal') {
            if(val.type == "http://www.w3.org/2001/XMLSchema#integer") {
                var tmp = parseInt(val.value);
                //if(isNaN(tmp)) {
```

```
        //    return false;
        //} else {
        return tmp;
        //}
    } else if(val.type == "http://www.w3.org/2001/XMLSchema#decimal") {
        var tmp = parseFloat(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
    } else if (val.type == "http://www.w3.org/2001/XMLSchema#float") {
        var tmp = parseFloat(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
    } else if (val.type == "http://www.w3.org/2001/XMLSchema#double") {
        var tmp = parseFloat(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
    } else if (val.type == "http://www.w3.org/2001/XMLSchema#nonPositiveInteger") {
        var tmp = parseFloat(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
    } else if (val.type == "http://www.w3.org/2001/XMLSchema#negativeInteger") {
        var tmp = parseInt(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
    } else if (val.type == "http://www.w3.org/2001/XMLSchema#long") {
        var tmp = parseInt(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
    } else if (val.type == "http://www.w3.org/2001/XMLSchema#int") {
        var tmp = parseInt(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
    } else if (val.type == "http://www.w3.org/2001/XMLSchema#short") {
        var tmp = parseInt(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
    } else if (val.type == "http://www.w3.org/2001/XMLSchema#byte") {
        var tmp = parseInt(val.value);
        //if(isNaN(tmp)) {
        //    return false;
        //} else {
        return tmp;
        //}
```

```
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#nonNegativeInteger") {
                var tmp = parseInt(val.value);
                //if(isNaN(tmp)) {
                //    return false;
                //} else {
                return tmp;
                //}
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#unsignedLong") {
                var tmp = parseInt(val.value);
                //if(isNaN(tmp)) {
                //    return false;
                //} else {
                return tmp;
                //}
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#unsignedInt") {
                var tmp = parseInt(val.value);
                //if(isNaN(tmp)) {
                //    return false;
                //} else {
                return tmp;
                //}
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#unsignedShort") {
                var tmp = parseInt(val.value);
                //if(isNaN(tmp)) {
                //    return false;
                //} else {
                return tmp;
                //}
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#unsignedByte") {
                var tmp = parseInt(val.value);
                //if(isNaN(tmp)) {
                //    return false;
                //} else {
                return tmp;
                //}
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#positiveInteger" ) {
                var tmp = parseInt(val.value);
                //if(isNaN(tmp)) {
                //    return false;
                //} else {
                return tmp;
                //}
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#date" ||
                val.type == "http://www.w3.org/2001/XMLSchema#dateTime" ) {
                try {
                    var d = TreeUtils.parseISO8601(val.value);
                    return(d);
                } catch(e) {
                    return null;
                }
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#boolean" ) {
                return val.value === true || val.value === 'true' || val.value === '1' || val.value === 1
|| val.value === true ? true :
                    val.value === false || val.value === 'false' || val.value === '0' || val.value === 0
|| val.value === false ? false :
                        undefined;
            } else if (val.type == "http://www.w3.org/2001/XMLSchema#string" ) {
                return val.value === null || val.value === undefined ? undefined : ''+val.value;
            } else if (val.type == null) {
                // plain literal -> just manipulate the string
                return val.value;
            } else {
                return val.value
            }
        } else {
            // @todo
            //console.log("not implemented yet");
```

```
            throw("value not supported in operations yet");
        }
    };

    /*
     A logical-or that encounters an error on only one branch will return TRUE if the other branch is
TRUE and an error if the other branch is FALSE.
     A logical-or or logical-and that encounters errors on both branches will produce either of the
errors.
     */
    QueryFilters.runOrFunction = function(filterExpr, bindings, queryEngine, dataset, env) {

        var acum = null;

        for(var i=0; i< filterExpr.operands.length; i++) {
            var ebv = QueryFilters.runFilter(filterExpr.operands[i], bindings, queryEngine, dataset, env);
            if(QueryFilters.isEbvError(ebv) == false) {
                ebv = QueryFilters.ebv(ebv);
            }

            if(acum == null) {
                acum = ebv;
            } else if(QueryFilters.isEbvError(ebv)) {
                if(QueryFilters.isEbvError(acum)) {
                    acum = QueryFilters.ebvError();
                } else if(acum === true) {
                    acum = true;
                } else {
                    acum = QueryFilters.ebvError();
                }
            } else if(ebv === true) {
                acum = true;
            } else {
                if(QueryFilters.isEbvError(acum)) {
                    acum = QueryFilters.ebvError();
                }
            }
        }

        return QueryFilters.ebvBoolean(acum);
    };

    /*
     A logical-and that encounters an error on only one branch will return an error if the other branch
is TRUE and FALSE if the other branch is FALSE.
     A logical-or or logical-and that encounters errors on both branches will produce either of the
errors.
     */
    QueryFilters.runAndFunction = function(filterExpr, bindings, queryEngine, dataset, env) {

        var acum = null;

        for(var i=0; i< filterExpr.operands.length; i++) {

            var ebv = QueryFilters.runFilter(filterExpr.operands[i], bindings, queryEngine, dataset, env);

            if(QueryFilters.isEbvError(ebv) == false) {
                ebv = QueryFilters.ebv(ebv);
            }

            if(acum == null) {
                acum = ebv;
            } else if(QueryFilters.isEbvError(ebv)) {
                if(QueryFilters.isEbvError(acum)) {
                    acum = QueryFilters.ebvError();
                } else if(acum === true) {
                    acum = QueryFilters.ebvError();
```

```
                } else {
                    acum = false;
                }
            } else if(ebv === true) {
                if(QueryFilters.isEbvError(acum)) {
                    acum = QueryFilters.ebvError();
                }
            } else {
                acum = false;
            }
        }

        return QueryFilters.ebvBoolean(acum);
    };


    QueryFilters.runEqualityFunction = function(op1, op2, bindings, queryEngine, dataset, env) {
        if(QueryFilters.isEbvError(op1) || QueryFilters.isEbvError(op2)) {
            return QueryFilters.ebvError();
        }
        if(QueryFilters.isNumeric(op1) && QueryFilters.isNumeric(op2)) {
            var eop1 = QueryFilters.effectiveTypeValue(op1);
            var eop2 = QueryFilters.effectiveTypeValue(op2);
            if(isNaN(eop1) || isNaN(eop2)) {
                return QueryFilters.ebvBoolean(QueryFilters.RDFTermEquality(op1, op2, queryEngine, env));
            } else {
                return QueryFilters.ebvBoolean(eop1 == eop2);
            }
        } else if((QueryFilters.isSimpleLiteral(op1) || QueryFilters.isXsdType("string", op1)) &&
            (QueryFilters.isSimpleLiteral(op2) || QueryFilters.isXsdType("string", op2))) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) ==
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("boolean", op1) && QueryFilters.isXsdType("boolean", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) ==
QueryFilters.effectiveTypeValue(op2));
        } else if((QueryFilters.isXsdType("dateTime", op1)||QueryFilters.isXsdType("date", op1)) &&
(QueryFilters.isXsdType("dateTime", op2)||QueryFilters.isXsdType("date", op2))) {
            if(QueryFilters.isXsdType("dateTime", op1) && QueryFilters.isXsdType("date", op2)) {
                return QueryFilters.ebvFalse();
            }
            if(QueryFilters.isXsdType("date", op1) && QueryFilters.isXsdType("dateTime", op2)) {
                return QueryFilters.ebvFalse();
            }

            var comp = TreeUtils.compareDateComponents(op1.value, op2.value);
            if(comp != null) {
                if(comp == 0) {
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else {
                return QueryFilters.ebvError();
            }
        } else if(QueryFilters.isRDFTerm(op1) && QueryFilters.isRDFTerm(op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.RDFTermEquality(op1, op2, queryEngine, env));
        } else {
            return QueryFilters.ebvFalse();
        }
    };

    QueryFilters.runGtFunction = function(op1, op2, bindings) {
        if(QueryFilters.isEbvError(op1) || QueryFilters.isEbvError(op2)) {
            return QueryFilters.ebvError();
        }

        if(QueryFilters.isNumeric(op1) && QueryFilters.isNumeric(op2)) {
```

```
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) >
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isSimpleLiteral(op1) && QueryFilters.isSimpleLiteral(op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) >
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("string", op1) && QueryFilters.isXsdType("string", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) >
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("boolean", op1) && QueryFilters.isXsdType("boolean", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) >
QueryFilters.effectiveTypeValue(op2));
        } else if((QueryFilters.isXsdType("dateTime", op1) || QueryFilters.isXsdType("date", op1)) &&
            (QueryFilters.isXsdType("dateTime", op2) || QueryFilters.isXsdType("date", op2))) {
            if(QueryFilters.isXsdType("dateTime", op1) && QueryFilters.isXsdType("date", op2)) {
                return QueryFilters.ebvFalse();
            }
            if(QueryFilters.isXsdType("date", op1) && QueryFilters.isXsdType("dateTime", op2)) {
                return QueryFilters.ebvFalse();
            }

            var comp = TreeUtils.compareDateComponents(op1.value, op2.value);
            if(comp != null) {
                if(comp == 1) {
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else {
                return QueryFilters.ebvError();
            }
        } else {
            return QueryFilters.ebvFalse();
        }
    };

    /**
     * Total gt function used when sorting bindings in the SORT BY clause.
     *
     * @todo
     * Some criteria are not clear
     */
    QueryFilters.runTotalGtFunction = function(op1,op2) {
        if(QueryFilters.isEbvError(op1) || QueryFilters.isEbvError(op2)) {
            return QueryFilters.ebvError();
        }

        if((QueryFilters.isNumeric(op1) && QueryFilters.isNumeric(op2)) ||
            (QueryFilters.isSimpleLiteral(op1) && QueryFilters.isSimpleLiteral(op2)) ||
            (QueryFilters.isXsdType("string",op1) && QueryFilters.isSimpleLiteral("string",op2)) ||
            (QueryFilters.isXsdType("boolean",op1) && QueryFilters.isSimpleLiteral("boolean",op2)) ||
            (QueryFilters.isXsdType("dateTime",op1) && QueryFilters.isSimpleLiteral("dateTime",op2))) {
            return QueryFilters.runGtFunction(op1, op2, []);
        } else if(op1.token && op1.token === 'uri' && op2.token && op2.token === 'uri') {
            return QueryFilters.ebvBoolean(op1.value > op2.value);
        } else if(op1.token && op1.token === 'literal' && op2.token && op2.token === 'literal') {
            // one of the literals must have type/lang and the othe may not have them
            return QueryFilters.ebvBoolean(""+op1.value+op1.type+op1.lang > ""+op2.value+op2.type
+op2.lang);
        } else if(op1.token && op1.token === 'blank' && op2.token && op2.token === 'blank') {
            return QueryFilters.ebvBoolean(op1.value > op2.value);
        } else if(op1.value && op2.value) {
            return QueryFilters.ebvBoolean(op1.value > op2.value);
        } else {
            return QueryFilters.ebvTrue();
        }
    };
```

```
    QueryFilters.runLtFunction = function(op1, op2, bindings) {
        if(QueryFilters.isEbvError(op1) || QueryFilters.isEbvError(op2)) {
            return QueryFilters.ebvError();
        }

        if(QueryFilters.isNumeric(op1) && QueryFilters.isNumeric(op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) <
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isSimpleLiteral(op1) && QueryFilters.isSimpleLiteral(op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) <
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("string", op1) && QueryFilters.isXsdType("string", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) <
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("boolean", op1) && QueryFilters.isXsdType("boolean", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) <
QueryFilters.effectiveTypeValue(op2));
        } else if((QueryFilters.isXsdType("dateTime", op1) || QueryFilters.isXsdType("date", op1)) &&
            (QueryFilters.isXsdType("dateTime", op2) || QueryFilters.isXsdType("date", op2))) {
            if(QueryFilters.isXsdType("dateTime", op1) && QueryFilters.isXsdType("date", op2)) {
                return QueryFilters.ebvFalse();
            }
            if(QueryFilters.isXsdType("date", op1) && QueryFilters.isXsdType("dateTime", op2)) {
                return QueryFilters.ebvFalse();
            }

            var comp = TreeUtils.compareDateComponents(op1.value, op2.value);
            if(comp != null) {
                if(comp == -1) {
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else {
                return QueryFilters.ebvError();
            }
        } else {
            return QueryFilters.ebvFalse();
        }
    };


    QueryFilters.runGtEqFunction = function(op1, op2, bindings) {
        if(QueryFilters.isEbvError(op1) || QueryFilters.isEbvError(op2)) {
            return QueryFilters.ebvError();
        }

        if(QueryFilters.isNumeric(op1) && QueryFilters.isNumeric(op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) >=
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isSimpleLiteral(op1) && QueryFilters.isSimpleLiteral(op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) >=
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("string", op1) && QueryFilters.isXsdType("string", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) >=
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("boolean", op1) && QueryFilters.isXsdType("boolean", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) >=
QueryFilters.effectiveTypeValue(op2));
        } else if((QueryFilters.isXsdType("dateTime", op1) || QueryFilters.isXsdType("date", op1)) &&
            (QueryFilters.isXsdType("dateTime", op2) || QueryFilters.isXsdType("date", op2))) {
            if(QueryFilters.isXsdType("dateTime", op1) && QueryFilters.isXsdType("date", op2)) {
                return QueryFilters.ebvFalse();
            }
            if(QueryFilters.isXsdType("date", op1) && QueryFilters.isXsdType("dateTime", op2)) {
                return QueryFilters.ebvFalse();
```

```
            }

            var comp = TreeUtils.compareDateComponents(op1.value, op2.value);
            if(comp != null) {
                if(comp != -1) {
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else {
                return QueryFilters.ebvError();
            }

        } else {
            return QueryFilters.ebvFalse();
        }
    };


    QueryFilters.runLtEqFunction = function(op1, op2, bindings) {
        if(QueryFilters.isEbvError(op1) || QueryFilters.isEbvError(op2)) {
            return QueryFilters.ebvError();
        }

        if(QueryFilters.isNumeric(op1) && QueryFilters.isNumeric(op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) <=
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isSimpleLiteral(op1) && QueryFilters.isSimpleLiteral(op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) <=
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("string", op1) && QueryFilters.isXsdType("string", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) <=
QueryFilters.effectiveTypeValue(op2));
        } else if(QueryFilters.isXsdType("boolean", op1) && QueryFilters.isXsdType("boolean", op2)) {
            return QueryFilters.ebvBoolean(QueryFilters.effectiveTypeValue(op1) <=
QueryFilters.effectiveTypeValue(op2));
        } else if((QueryFilters.isXsdType("dateTime", op1) || QueryFilters.isXsdType("date", op1)) &&
            (QueryFilters.isXsdType("dateTime", op2) || QueryFilters.isXsdType("date", op2))) {
            if(QueryFilters.isXsdType("dateTime", op1) && QueryFilters.isXsdType("date", op2)) {
                return QueryFilters.ebvFalse();
            }
            if(QueryFilters.isXsdType("date", op1) && QueryFilters.isXsdType("dateTime", op2)) {
                return QueryFilters.ebvFalse();
            }

            var comp = TreeUtils.compareDateComponents(op1.value, op2.value);
            if(comp != null) {
                if(comp != 1) {
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else {
                return QueryFilters.ebvError();
            }
        } else {
            return QueryFilters.ebvFalse();
        }
    };

    QueryFilters.runAddition = function(summand, summands, bindings, queryEngine, dataset, env) {
        var summandOp = QueryFilters.runFilter(summand,bindings,queryEngine, dataset, env);
        if(QueryFilters.isEbvError(summandOp)) {
            return QueryFilters.ebvError();
        }

        var acum = summandOp;
```

```
        if(QueryFilters.isNumeric(summandOp)) {
            for(var i=0; i<summands.length; i++) {
                var nextSummandOp = QueryFilters.runFilter(summands[i].expression, bindings,queryEngine,
dataset, env);
                if(QueryFilters.isNumeric(nextSummandOp)) {
                    if(summands[i].operator === '+') {
                        acum = QueryFilters.runSumFunction(acum, nextSummandOp);
                    } else if(summands[i].operator === '-') {
                        acum = QueryFilters.runSubFunction(acum, nextSummandOp);
                    }
                } else {
                    return QueryFilters.ebvFalse();
                }
            }
            return acum;
        } else {
            return QueryFilters.ebvFalse();
        }
    };

    QueryFilters.runSumFunction = function(suma, sumb) {
        if(QueryFilters.isEbvError(suma) || QueryFilters.isEbvError(sumb)) {
            return QueryFilters.ebvError();
        }
        var val = QueryFilters.effectiveTypeValue(suma) + QueryFilters.effectiveTypeValue(sumb);
        if(QueryFilters.isDouble(suma) || QueryFilters.isDouble(sumb)) {
            return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#double", value:val};
        } else if(QueryFilters.isFloat(suma) || QueryFilters.isFloat(sumb)) {
            return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#float", value:val};
        } else if(QueryFilters.isDecimal(suma) || QueryFilters.isDecimal(sumb)) {
            return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#decimal", value:val};
        } else {
            return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#integer", value:val};
        }
    };
    QueryFilters.runSubFunction = function(suma, sumb) {
        if(QueryFilters.isEbvError(suma) || QueryFilters.isEbvError(sumb)) {
            return QueryFilters.ebvError();
        }
        var val = QueryFilters.effectiveTypeValue(suma) - QueryFilters.effectiveTypeValue(sumb);

        if(QueryFilters.isDouble(suma) || QueryFilters.isDouble(sumb)) {
            return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#double", value:val};
        } else if(QueryFilters.isFloat(suma) || QueryFilters.isFloat(sumb)) {
            return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#float", value:val};
        } else if(QueryFilters.isDecimal(suma) || QueryFilters.isDecimal(sumb)) {
            return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#decimal", value:val};
        } else {
            return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#integer", value:val};
        }
    };

    QueryFilters.runMultiplication = function(factor, factors, bindings, queryEngine, dataset, env) {
        var factorOp = QueryFilters.runFilter(factor,bindings,queryEngine, dataset, env);
        if(QueryFilters.isEbvError(factorOp)) {
            return factorOp;
        }

        var acum = factorOp;
        if(QueryFilters.isNumeric(factorOp)) {
            for(var i=0; i<factors.length; i++) {
                var nextFactorOp = QueryFilters.runFilter(factors[i].expression, bindings,queryEngine,
dataset, env);
                if(QueryFilters.isEbvError(nextFactorOp)) {
                    return factorOp;
                }
                if(QueryFilters.isNumeric(nextFactorOp)) {
```

```
                if(factors[i].operator === '*') {
                    acum = QueryFilters.runMulFunction(acum, nextFactorOp);
                } else if(factors[i].operator === '/') {
                    acum = QueryFilters.runDivFunction(acum, nextFactorOp);
                }
            } else {
                return QueryFilters.ebvFalse();
            }
        }
        return acum;
    } else {
        return QueryFilters.ebvFalse();
    }
};

QueryFilters.runMulFunction = function(faca, facb) {
    if(QueryFilters.isEbvError(faca) || QueryFilters.isEbvError(facb)) {
        return QueryFilters.ebvError();
    }
    var val = QueryFilters.effectiveTypeValue(faca) * QueryFilters.effectiveTypeValue(facb);

    if(QueryFilters.isDouble(faca) || QueryFilters.isDouble(facb)) {
        return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#double", value:val};
    } else if(QueryFilters.isFloat(faca) || QueryFilters.isFloat(facb)) {
        return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#float", value:val};
    } else if(QueryFilters.isDecimal(faca) || QueryFilters.isDecimal(facb)) {
        return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#decimal", value:val};
    } else {
        return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#integer", value:val};
    }
};

QueryFilters.runDivFunction = function(faca, facb) {
    if(QueryFilters.isEbvError(faca) || QueryFilters.isEbvError(facb)) {
        return QueryFilters.ebvError();
    }
    var val = QueryFilters.effectiveTypeValue(faca) / QueryFilters.effectiveTypeValue(facb);

    if(QueryFilters.isDouble(faca) || QueryFilters.isDouble(facb)) {
        return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#double", value:val};
    } else if(QueryFilters.isFloat(faca) || QueryFilters.isFloat(facb)) {
        return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#float", value:val};
    } else if(QueryFilters.isDecimal(faca) || QueryFilters.isDecimal(facb)) {
        return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#decimal", value:val};
    } else {
        return {token: 'literal', type:"http://www.w3.org/2001/XMLSchema#integer", value:val};
    }
};

QueryFilters.runBuiltInCall = function(builtincall, args, bindings, queryEngine, dataset, env) {
    if(builtincall === 'notexists' || builtincall === 'exists') {
        // Run the query in the filter applying bindings

        var cloned = JSON.parse(JSON.stringify(args[0])); // @todo CHANGE THIS!!
        var ast = queryEngine.abstractQueryTree.parseSelect({pattern:cloned}, bindings);
        ast = queryEngine.abstractQueryTree.bind(ast.pattern, bindings);

        var result = queryEngine.executeSelectUnit([ {kind:'*'} ],
            dataset,
            ast,
            env);

        if(builtincall === 'exists') {
            return QueryFilters.ebvBoolean(result.length!==0);
        } else {
            return QueryFilters.ebvBoolean(result.length===0);
        }
```

```
        }  else {

            var ops = [];
            for(var i=0; i<args.length; i++) {
                if(args[i].token === 'var') {
                    ops.push(args[i]);
                } else {
                    var op = QueryFilters.runFilter(args[i], bindings, queryEngine, dataset, env);
                    if(QueryFilters.isEbvError(op)) {
                        return op;
                    }
                    ops.push(op);
                }
            }

            if(builtincall === 'str') {
                if(ops[0].token === 'literal') {
                    // lexical form literals
                    return {token: 'literal', type:null, value:""+ops[0].value}; // type null? or "http://
www.w3.org/2001/XMLSchema#string"
                } else if(ops[0].token === 'uri'){
                    // codepoint URIs
                    return {token: 'literal', type:null, value:ops[0].value}; // idem
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else if(builtincall === 'lang') {
                if(ops[0].token === 'literal'){
                    if(ops[0].lang != null) {
                        return {token: 'literal', value:""+ops[0].lang};
                    } else {
                        return {token: 'literal', value:""};
                    }
                } else {
                    return QueryFilters.ebvError();
                }
            } else if(builtincall === 'datatype') {
                if(ops[0].token === 'literal'){
                    var lit = ops[0];
                    if(lit.type != null) {
                        if(typeof(lit.type) === 'string') {
                            return {token: 'uri', value:lit.type, prefix:null, suffix:null};
                        } else {
                            return lit.type;
                        }
                    } else if(lit.lang == null) {
                        return {token: 'uri', value:'http://www.w3.org/2001/XMLSchema#string',
prefix:null, suffix:null};
                    } else {
                        return QueryFilters.ebvError();
                    }
                } else {
                    return QueryFilters.ebvError();
                }
            } else if(builtincall === 'isliteral') {
                if(ops[0].token === 'literal'){
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else if(builtincall === 'isblank') {
                if(ops[0].token === 'blank'){
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
```

```
            } else if(builtincall === 'isuri' || builtincall === 'isiri') {
                if(ops[0].token === 'uri'){
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else if(builtincall === 'sameterm') {
                var op1 = ops[0];
                var op2 = ops[1];
                var res = QueryFilters.RDFTermEquality(op1, op2, queryEngine, env);
                if(QueryFilters.isEbvError(res)) {
                    res = false;
                }
                return QueryFilters.ebvBoolean(res);
            } else if(builtincall === 'langmatches') {
                var lang = ops[0];
                var langRange = ops[1];

                if(lang.token === 'literal' && langRange.token === 'literal'){
                    if(langRange.value === '*' && lang.value != '') {
                        return QueryFilters.ebvTrue();
                    } else {
                        return QueryFilters.ebvBoolean(lang.value.toLowerCase().indexOf
(langRange.value.toLowerCase()) === 0)
                    }
                } else {
                    return QueryFilters.ebvError();
                }
            } else if(builtincall === 'bound') {
                var boundVar = ops[0].value;
                var acum = [];
                if(boundVar == null) {
                    return QueryFilters.ebvError();
                } else  if(bindings[boundVar] != null) {
                    return QueryFilters.ebvTrue();
                } else {
                    return QueryFilters.ebvFalse();
                }
            } else {
                throw ("Builtin call "+builtincall+" not implemented yet");
            }
        }
    };

    QueryFilters.runUnaryExpression = function(unaryexpression, expression, bindings, queryEngine,
dataset, env) {
        var op = QueryFilters.runFilter(expression, bindings,queryEngine, dataset, env);
        if(QueryFilters.isEbvError(op)) {
            return op;
        }

        if(unaryexpression === '!') {
            var res = QueryFilters.ebv(op);
            //console.log("** Unary ! ");
            //console.log(op)
            if(QueryFilters.isEbvError(res)) {
                //console.log("--- ERROR")
                //console.log(QueryFilters.ebvFalse())
                //console.log("\r\n")

                // ??
                return QueryFilters.ebvFalse();
            } else {
                res = !res;
                //console.log("--- BOOL")
                //console.log(QueryFilters.ebvBoolean(res))
                //console.log("\r\n")
```

```
                return QueryFilters.ebvBoolean(res);
            }
        } else if(unaryexpression === '+') {
            if(QueryFilters.isNumeric(op)) {
                return op;
            } else {
                return QueryFilters.ebvError();
            }
        } else if(unaryexpression === '-') {
            if(QueryFilters.isNumeric(op)) {
                var clone = {};
                for(var p in op) {
                    clone[p] = op[p];
                }
                clone.value = -clone.value;
                return clone;
            } else {
                return QueryFilters.ebvError();
            }
        }
    };

    QueryFilters.runRegex = function(text, pattern, flags, bindings, queryEngine, dataset, env) {

        if(text != null) {
            text = QueryFilters.runFilter(text, bindings, queryEngine, dataset, env);
        } else {
            return QueryFilters.ebvError();
        }

        if(pattern != null) {
            pattern = QueryFilters.runFilter(pattern, bindings, queryEngine, dataset, env);
        } else {
            return QueryFilters.ebvError();
        }

        if(flags != null) {
            flags = QueryFilters.runFilter(flags, bindings, queryEngine, dataset, env);
        }


        if(pattern != null && pattern.token === 'literal' && (flags == null || flags.token ===
'literal')) {
            pattern = pattern.value;
            flags = (flags == null) ? null : flags.value;
        } else {
            return QueryFilters.ebvError();
        }

        if(text!= null && text.token == 'var') {
            if(bindings[text.value] != null) {
                text = bindings[text.value];
            } else {
                return QueryFilters.ebvError();
            }
        } else if(text!=null && text.token === 'literal') {
            if(text.type == null || QueryFilters.isXsdType("string",text)) {
                text = text.value
            } else {
                return QueryFilters.ebvError();
            }
        } else {
            return QueryFilters.ebvError();
        }

        var regex;
```

```
        if(flags == null) {
            regex = new RegExp(pattern);
        } else {
            regex = new RegExp(pattern,flags.toLowerCase());
        }
        if(regex.exec(text)) {
            return QueryFilters.ebvTrue();
        } else {
            return QueryFilters.ebvFalse();
        }
    };

    QueryFilters.normalizeLiteralDatatype = function(literal, queryEngine, env) {
        if(literal.value.type == null || typeof(literal.value.type) != 'object') {
            return literal;
        } else {
            // type can be parsed as a hash using namespaces
            literal.value.type =  TreeUtils.lexicalFormBaseUri(literal.value.type, env);
            return literal;
        }
    };

    QueryFilters.runIriRefOrFunction = function(iriref, args, bindings,queryEngine, dataset, env) {
        if(args == null) {
            return iriref;
        } else {
            var ops = [];
            for(var i=0; i<args.length; i++) {
                ops.push(QueryFilters.runFilter(args[i], bindings, queryEngine, dataset, env))
            }

            var fun = TreeUtils.lexicalFormBaseUri(iriref, env);

            if(fun == "http://www.w3.org/2001/XMLSchema#integer" ||
                fun == "http://www.w3.org/2001/XMLSchema#decimal" ||
                fun == "http://www.w3.org/2001/XMLSchema#double" ||
                fun == "http://www.w3.org/2001/XMLSchema#nonPositiveInteger" ||
                fun == "http://www.w3.org/2001/XMLSchema#negativeInteger" ||
                fun == "http://www.w3.org/2001/XMLSchema#long" ||
                fun == "http://www.w3.org/2001/XMLSchema#int" ||
                fun == "http://www.w3.org/2001/XMLSchema#short" ||
                fun == "http://www.w3.org/2001/XMLSchema#byte" ||
                fun == "http://www.w3.org/2001/XMLSchema#nonNegativeInteger" ||
                fun == "http://www.w3.org/2001/XMLSchema#unsignedLong" ||
                fun == "http://www.w3.org/2001/XMLSchema#unsignedInt" ||
                fun == "http://www.w3.org/2001/XMLSchema#unsignedShort" ||
                fun == "http://www.w3.org/2001/XMLSchema#unsignedByte" ||
                fun == "http://www.w3.org/2001/XMLSchema#positiveInteger") {
                var from = ops[0];
                if(from.token === 'literal') {
                    from = QueryFilters.normalizeLiteralDatatype(from, queryEngine, env);
                    if(from.type == "http://www.w3.org/2001/XMLSchema#integer" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#decimal" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#double" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#nonPositiveInteger" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#negativeInteger" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#long" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#int" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#short" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#byte" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#nonNegativeInteger" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#unsignedLong" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#unsignedInt" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#unsignedShort" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#unsignedByte" ||
                        from.type == "http://www.w3.org/2001/XMLSchema#positiveInteger") {
                        from.type = fun;
```

```
                    return from;
                } else if(from.type == 'http://www.w3.org/2001/XMLSchema#boolean') {
                    if(QueryFilters.ebv(from) == true) {
                        from.type = fun;
                        from.value = 1;
                    } else {
                        from.type = fun;
                        from.value = 0;
                    }
                    return from;
                } else if(from.type == 'http://www.w3.org/2001/XMLSchema#float' ||
                    from.type == 'http://www.w3.org/2001/XMLSchema#double') {
                    from.type = fun;
                    from.value = parseInt(from.value);
                    return from;
                } else if(from.type == 'http://www.w3.org/2001/XMLSchema#string' || from.type ==
null) {
                    if(from.value.split(".").length > 2) {
                        return QueryFilters.ebvError();
                    } else if (from.value.split("-").length > 2) {
                        return QueryFilters.ebvError();
                    } else if (from.value.split("/").length > 2) {
                        return QueryFilters.ebvError();
                    } else if (from.value.split("+").length > 2) {
                        return QueryFilters.ebvError();
                    }

                    // @todo improve this with regular expressions for each lexical representation
                    if(fun == "http://www.w3.org/2001/XMLSchema#decimal") {
                        if(from.value.indexOf("e") != -1 || from.value.indexOf("E") != -1) {
                            return QueryFilters.ebvError();
                        }
                    }

                    // @todo improve this with regular expressions for each lexical representation
                    if(fun == "http://www.w3.org/2001/XMLSchema#int" || fun == "http://
www.w3.org/2001/XMLSchema#integer") {
                        if(from.value.indexOf("e") != -1 || from.value.indexOf("E") != -1 ||
from.value.indexOf(".") != -1) {
                            return QueryFilters.ebvError();
                        }
                    }

                    try {
                        from.value = parseInt(parseFloat(from.value));
                        if(isNaN(from.value)) {
                            return QueryFilters.ebvError();
                        } else {
                            from.type = fun;
                            return from;
                        }
                    } catch(e) {
                        return QueryFilters.ebvError();
                    }
                } else {
                    return QueryFilters.ebvError();
                }
            } else {
                return QueryFilters.ebvError();
            }
        } else if(fun == "http://www.w3.org/2001/XMLSchema#boolean") {
            var from = ops[0];
            if(from.token === "literal" && from.type == null) {
                if(from.value === "true" || from.value === "1") {
                    return QueryFilters.ebvTrue();
                } else if(from.value === "false" || from.value === "0" ) {
                    return QueryFilters.ebvFalse();
```

```
            } else {
                return QueryFilters.ebvError();
            }
        } else if(from.token === "literal") {
            if(QueryFilters.isEbvError(from)) {
                return from;
            } else {
                return QueryFilters.ebvBoolean(from);
            }
        } else {
            return QueryFilters.ebvError();
        }
    } else if(fun == "http://www.w3.org/2001/XMLSchema#string") {
        var from = ops[0];
        if(from.token === 'literal') {
            from = QueryFilters.normalizeLiteralDatatype(from, queryEngine, env);
            if(from.type == "http://www.w3.org/2001/XMLSchema#integer" ||
                from.type == "http://www.w3.org/2001/XMLSchema#decimal" ||
                from.type == "http://www.w3.org/2001/XMLSchema#double" ||
                from.type == "http://www.w3.org/2001/XMLSchema#nonPositiveInteger" ||
                from.type == "http://www.w3.org/2001/XMLSchema#negativeInteger" ||
                from.type == "http://www.w3.org/2001/XMLSchema#long" ||
                from.type == "http://www.w3.org/2001/XMLSchema#int" ||
                from.type == "http://www.w3.org/2001/XMLSchema#short" ||
                from.type == "http://www.w3.org/2001/XMLSchema#byte" ||
                from.type == "http://www.w3.org/2001/XMLSchema#nonNegativeInteger" ||
                from.type == "http://www.w3.org/2001/XMLSchema#unsignedLong" ||
                from.type == "http://www.w3.org/2001/XMLSchema#unsignedInt" ||
                from.type == "http://www.w3.org/2001/XMLSchema#unsignedShort" ||
                from.type == "http://www.w3.org/2001/XMLSchema#unsignedByte" ||
                from.type == "http://www.w3.org/2001/XMLSchema#positiveInteger" ||
                from.type == "http://www.w3.org/2001/XMLSchema#float") {
                from.type = fun;
                from.value = ""+from.value;
                return from;
            } else if(from.type == "http://www.w3.org/2001/XMLSchema#string") {
                return from;
            } else if(from.type == "http://www.w3.org/2001/XMLSchema#boolean") {
                if(QueryFilters.ebv(from)) {
                    from.type = fun;
                    from.value = 'true';
                } else {
                    from.type = fun;
                    from.value = 'false';
                }
                return from;
            } else if(from.type == "http://www.w3.org/2001/XMLSchema#dateTime" ||
                from.type == "http://www.w3.org/2001/XMLSchema#date") {
                from.type = fun;
                if(typeof(from.value) != 'string') {
                    from.value = TreeUtils.iso8601(from.value);
                }
                return from;
            } else if(from.type == null) {
                from.value = ""+from.value;
                from.type = fun;
                return from;
            } else {
                return QueryFilters.ebvError();
            }
        } else if(from.token === 'uri') {
            return {token: 'literal',
                value: TreeUtils.lexicalFormBaseUri(from, env),
                type: fun,
                lang: null};
        } else {
            return QueryFilters.ebvError();
```

```
                }
            } else if(fun == "http://www.w3.org/2001/XMLSchema#dateTime" || fun == "http://
www.w3.org/2001/XMLSchema#date") {
                from = ops[0];
                if(from.type == "http://www.w3.org/2001/XMLSchema#dateTime" || from.type == "http://
www.w3.org/2001/XMLSchema#date") {
                    return from;
                } else if(from.type == "http://www.w3.org/2001/XMLSchema#string" || from.type == null) {
                    try {
                        from.value = TreeUtils.iso8601(Utils.parseStrictISO8601(from.value));
                        from.type = fun;
                        return from;
                    } catch(e) {
                        return QueryFilters.ebvError();
                    }
                } else {
                    return QueryFilters.ebvError();
                }
            } else if(fun == "http://www.w3.org/2001/XMLSchema#float") {
                var from = ops[0];
                if(from.token === 'literal') {
                    from = QueryFilters.normalizeLiteralDatatype(from, queryEngine, env);
                    if(from.type == 'http://www.w3.org/2001/XMLSchema#decimal' ||
                        from.type == 'http://www.w3.org/2001/XMLSchema#int') {
                        from.type = fun;
                        from.value = parseFloat(from.value);
                        return from;
                    } else if(from.type == 'http://www.w3.org/2001/XMLSchema#boolean') {
                        if(QueryFilters.ebv(from) == true) {
                            from.type = fun;
                            from.value = 1.0;
                        } else {
                            from.type = fun;
                            from.value = 0.0;
                        }
                        return from;
                    } else if(from.type == 'http://www.w3.org/2001/XMLSchema#float' ||
                        from.type == 'http://www.w3.org/2001/XMLSchema#double') {
                        from.type = fun;
                        from.value = parseFloat(from.value);
                        return from;
                    } else if(from.type == 'http://www.w3.org/2001/XMLSchema#string') {
                        try {
                            from.value = parseFloat(from.value);
                            if(isNaN(from.value)) {
                                return QueryFilters.ebvError();
                            } else {
                                from.type = fun;
                                return from;
                            }
                        } catch(e) {
                            return QueryFilters.ebvError();
                        }
                    } else if(from.type == null) {
                        // checking some exceptions that are parsed as Floats by JS
                        if(from.value.split(".").length > 2) {
                            return QueryFilters.ebvError();
                        } else if (from.value.split("-").length > 2) {
                            return QueryFilters.ebvError();
                        } else if (from.value.split("/").length > 2) {
                            return QueryFilters.ebvError();
                        } else if (from.value.split("+").length > 2) {
                            return QueryFilters.ebvError();
                        }
                        try {
                            from.value = parseFloat(from.value);
                            if(isNaN(from.value)) {
```

```
                            return QueryFilters.ebvError();
                        } else {
                            from.type = fun;
                            return from;
                        }
                    } catch(e) {
                        return QueryFilters.ebvError();
                    }
                } else {
                    return QueryFilters.ebvError();
                }
            } else {
                return QueryFilters.ebvError();
            }
        } else {
            // unknown function
            return QueryFilters.ebvError();
        }
    }
};
return QueryFilters;
});
define([], function () {
    var QueryPlanDPSize = {};

    QueryPlanDPSize.variablesInBGP = function(bgp) {
        // may be cached in the pattern
        var variables = bgp.variables;
        if(variables) {
            return variables;
        }

        var components =  bgp.value || bgp;
        variables  = [];
        for(var comp in components) {
            if(components[comp] && components[comp].token === "var") {
                variables.push(components[comp].value);
            } else if(components[comp] && components[comp].token === "blank") {
                variables.push("blank:"+components[comp].value);
            }
        }
        bgp.variables = variables;

        return variables;
    };

    QueryPlanDPSize.connected = function(leftPlan, rightPlan) {
        var varsLeft ="/"+leftPlan.vars.join("/")+"/";
        for(var i=0; i<rightPlan.vars.length; i++) {
            if(varsLeft.indexOf("/"+rightPlan.vars[i]+"/") != -1) {
                return true;
            }
        }

        return false;
    };

    QueryPlanDPSize.variablesIntersectionBGP = function(bgpa, bgpb) {
        var varsa = QueryPlanDPSize.variablesInBGP(bgpa).sort();
        var varsb = QueryPlanDPSize.variablesInBGP(bgpb).sort();
        var ia = 0;
        var ib = 0;

        var intersection = [];

        while(ia<varsa.length && ib<varsb.length) {
            if(varsa[ia] === varsb[ib]) {
```

```
            intersection.push(varsa[ia]);
            ia++;
            ib++;
        } else if(varsa[ia] < varsb[ib]) {
            ia++;
        } else {
            ib++;
        }
    }

    return intersection;
};

/**
 * All BGPs sharing variables are grouped together.
 */
QueryPlanDPSize.executeAndBGPsGroups = function(bgps) {
    var groups = {};
    var groupVars = {};
    var groupId = 0;

    for(var i=0; i<bgps.length; i++) {
        var bgp = bgps[i];
        var newGroups = {};
        var newGroupVars = {};

        var vars = [];
        for(var comp in bgp) {
            if(comp != '_cost') {
                if(bgp[comp].token === 'var') {
                    vars.push(bgp[comp].value);
                } else if(bgp[comp].token === 'blank') {
                    vars.push(bgp[comp].value);
                }
            }
        }


        var foundGroup = false;
        var currentGroupId = null;
        var toDelete = [];
        var toJoin = {};

        for(var nextGroupId in groupVars) {
            var groupVar = groupVars[nextGroupId];
            foundGroup = false;
            for(var j=0; j<vars.length; j++) {
                var thisVar = "/"+vars[j]+"/";
                if(groupVar.indexOf(thisVar) != -1) {
                    foundGroup = true;
                    break;
                }
            }

            if(foundGroup) {
                toJoin[nextGroupId] = true;
            } else {
                newGroups[nextGroupId] = groups[nextGroupId];
                newGroupVars[nextGroupId] = groupVars[nextGroupId];
            }
        }

        if(!foundGroup) {
            newGroups[groupId] = [bgp];
            newGroupVars[groupId] = "/"+(vars.join("/"))+"/";
            groupId++;
        } else {
```

```
            var acumGroups = [];
            var acumId = "";
            var acumVars = "";
            for(var gid in toJoin) {
                acumId = acumId+gid;
                acumGroups = acumGroups.concat(groups[gid]);
                acumVars = groupVars[gid];
            }

            acumVars = acumVars + vars.join("/") + "/";
            acumGroups.push(bgp);

            newGroups[acumId] = acumGroups;
            newGroupVars[acumId] = acumVars;
        }

        groups = newGroups;
        groupVars = newGroupVars;
    }

    var acum = [];
    for(var groupId in groups) {
        acum.push(groups[groupId]);
    }

    return acum;
};

QueryPlanDPSize.intersectionSize = function(leftPlan, rightPlan) {
    var idsRight = rightPlan.i.split("_");
    for(var i=0; i<idsRight.length; i++) {
        if(idsRight[i]=="")
            continue;
        if(leftPlan.i.indexOf('_'+idsRight[i]+'_') != -1) {
            return 1; // we just need to know if this value is >0
        }
    }
    return 0;
};

QueryPlanDPSize.createJoinTree = function(leftPlan, rightPlan) {
    var varsLeft ="/"+leftPlan.vars.join("/")+"/";
    var acumVars = leftPlan.vars.concat([]);
    var join = [];

    for(var i=0; i<rightPlan.vars.length; i++) {
        if(varsLeft.indexOf("/"+rightPlan.vars[i]+"/") != -1) {
            if(rightPlan.vars[i].indexOf("_:") == 0) {
                join.push("blank:"+rightPlan.vars[i]);
            } else {
                join.push(rightPlan.vars[i]);
            }
        } else {
            acumVars.push(rightPlan.vars[i]);
        }
    }

    var rightIds = rightPlan.i.split("_");
    var leftIds = leftPlan.i.split("_");
    var distinct = {};
    for(var i=0; i<rightIds.length; i++) {
        if(rightIds[i] != "") {
            distinct[rightIds[i]] = true;
        }
    }
    for(var i=0; i<leftIds.length; i++) {
        if(leftIds[i] != "") {
```

```
                distinct[leftIds[i]] = true;
            }
        }
        var ids = [];
        for(var id in distinct) {
            ids.push(id);
        }

        // new join tree
        return {
            left: leftPlan,
            right: rightPlan,
            cost: leftPlan.cost+rightPlan.cost,
            i: "_"+(ids.sort().join("_"))+"_",
            vars: acumVars,
            join: join
        };
    };

    QueryPlanDPSize.executeBushyTree = function(treeNode, dataset, queryEngine, env) {
        if(treeNode.left == null ) {
            return QueryPlanDPSize.executeEmptyJoinBGP(treeNode.right, dataset, queryEngine, env);
        } else if(treeNode.right == null) {
            return QueryPlanDPSize.executeEmptyJoinBGP(treeNode.left, dataset, queryEngine, env);
        } else {
            var resultsLeft = QueryPlanDPSize.executeBushyTree(treeNode.left, dataset, queryEngine, env);

            if(resultsLeft!=null) {
                var resultsRight = QueryPlanDPSize.executeBushyTree(treeNode.right, dataset, queryEngine,
env);
                if(resultsRight!=null) {
                    return QueryPlanDPSize.joinBindings2(treeNode.join, resultsLeft, resultsRight);
                } else {
                    return null;
                }
            }
        }
    };


    QueryPlanDPSize.executeAndBGPsDPSize = function(allBgps, dataset, queryEngine, env) {
        var groups = QueryPlanDPSize.executeAndBGPsGroups(allBgps);
        var groupResults = [];
        for(var g=0; g<groups.length; g++) {

            // Build bushy tree for this group
            var bgps = groups[g];
            var costFactor = 1;

            var bgpas = queryEngine.computeCosts(bgps,env);

            var bestPlans = {};
            var plans = {};
            var sizes = {};

            var maxSize = 1;
            var maxPlan = null;

            var cache = {};

            sizes['1'] = [];

            // Building plans of size 1
            for(var i=0; i<bgps.length; i++) {
                var vars = [];
                for(var comp in bgps[i]) {
                    if(comp != '_cost') {
```

```
                        if(bgps[i][comp].token === 'var') {
                            vars.push(bgps[i][comp].value);
                        } else if(bgps[i][comp].token === 'blank') {
                            vars.push(bgps[i][comp].value);
                        }
                    }
                }

                plans["_"+i+"_"] = {left: bgps[i], right:null, cost:bgps[i]._cost, i:('_'+i+'_'),
vars:vars};
                var plan = {left: bgps[i], right:null, cost:bgps[i]._cost, i:('_'+i+'_'), vars:vars};
                bestPlans["_"+i+"_"] = plan;
                delete bgps[i]['_cost'];
                cache["_"+i+"_"] = true;
                sizes['1'].push("_"+i+"_");
                if(maxPlan == null || maxPlan.cost>plan.cost) {
                    maxPlan = plan;
                }
            }

            // dynamic programming -> build plans of increasing size
            for(var s=2; s<=bgps.length; s++) { // size
                for(var sl=1; sl<s; sl++) { // size left plan
                    var sr = s - sl; // size right plan
                    var leftPlans = sizes[''+sl] || [];
                    var rightPlans = sizes[''+sr] || [];

                    for(var i=0; i<leftPlans.length; i++) {
                        for(var j=0; j<rightPlans.length; j++) {
                            if(leftPlans[i]===rightPlans[j])
                                continue;
                            var leftPlan = plans[leftPlans[i]];
                            var rightPlan = plans[rightPlans[j]];

                            // condition (1)
                            if(QueryPlanDPSize.intersectionSize(leftPlan, rightPlan) == 0) {
                                // condition (2)

                                if(QueryPlanDPSize.connected(leftPlan,rightPlan)) {
                                    maxSize = s;
                                    var p1 = bestPlans[leftPlan.i];  //QueryPlanDPSize.bestPlan(leftPlan,
bestPlans);
                                    var p2 = bestPlans[rightPlan.i]; //QueryPlanDPSize.bestPlan
(rightPlan, bestPlans);

                                    var currPlan = QueryPlanDPSize.createJoinTree(p1,p2);
                                    if(!cache[currPlan.i]) {
                                        cache[currPlan.i] = true;

                                        var costUnion = currPlan.cost+1;
                                        if(bestPlans[currPlan.i] != null) {
                                            costUnion = bestPlans[currPlan.i].cost;
                                        }

                                        var acum = sizes[s] || [];
                                        acum.push(currPlan.i);
                                        plans[currPlan.i] = currPlan;
                                        sizes[s] = acum;

                                        if(costUnion > currPlan.cost) {
                                            if(maxSize === s) {
                                                maxPlan = currPlan;
                                            }
                                            bestPlans[currPlan.i] = currPlan;
                                        }
                                    }
                                }
```

```
                        }
                    }
                }
            }
        }

        groupResults.push(maxPlan);
    }


    // now execute the Bushy trees and perform
    // cross products between groups
    var acum = null;

    for(var g=0; g<groupResults.length; g++) {
        var tree = groupResults[g];

        var result = QueryPlanDPSize.executeBushyTree(tree, dataset, queryEngine, env);
        if(acum == null) {
            acum = result;
        } else {
            acum = QueryPlanDPSize.crossProductBindings(acum, result);
        }
    };

    return acum;
};

QueryPlanDPSize.executeEmptyJoinBGP = function(bgp, dataset, queryEngine, queryEnv) {
    return QueryPlanDPSize.executeBGPDatasets(bgp, dataset, queryEngine, queryEnv);
};


QueryPlanDPSize.executeBGPDatasets = function(bgp, dataset, queryEngine, queryEnv) {
    // avoid duplicate queries in the same graph
    // merge of graphs is not guaranted here.
    var duplicates = {};

    if(bgp.graph == null) {
        //union through all default graph(s)
        var acum = [];
        for(var i=0; i<dataset.implicit.length; i++) {
            if(duplicates[dataset.implicit[i].oid] == null) {
                duplicates[dataset.implicit[i].oid] = true;
                bgp.graph = dataset.implicit[i];//.oid
                var results = queryEngine.rangeQuery(bgp, queryEnv);
                results = QueryPlanDPSize.buildBindingsFromRange(results, bgp);
                acum.push(results);
            }
        }
        var acumBindings = QueryPlanDPSize.unionManyBindings(acum);
        return acumBindings;
    } else if(bgp.graph.token === 'var') {
        // union through all named datasets
        var graphVar = bgp.graph.value;
        var acum = [];

        for(var i=0; i<dataset.named.length; i++) {
            if(duplicates[dataset.named[i].oid] == null) {
                duplicates[dataset.named[i].oid] = true;
                bgp.graph = dataset.named[i];//.oid

                var results = queryEngine.rangeQuery(bgp, queryEnv);
                if(results != null) {
                    results = QueryPlanDPSize.buildBindingsFromRange(results, bgp);
                    // add the graph bound variable to the result
                    for(var j=0; j< results.length; j++) {
```

```
                        results[j][graphVar] = dataset.named[i].oid;
                    }
                    acum.push(results);
                } else {
                    return null;
                }
            }
        }

        var acumBindings = QueryPlanDPSize.unionManyBindings(acum||[]);
        return acumBindings;

    } else {
        // graph already has an active value, just match.
        // Filtering the results will still be necessary
        var results = queryEngine.rangeQuery(bgp, queryEnv);
        if(results!=null) {
            results = QueryPlanDPSize.buildBindingsFromRange(results, bgp);
            return results;
        } else {
            return null;
        }
    }
};

QueryPlanDPSize.buildBindingsFromRange = function(results, bgp) {
    var variables = QueryPlanDPSize.variablesInBGP(bgp);
    var bindings = {};

    var components =  bgp.value||bgp;
    var bindings = {};
    for(comp in components) {
        if(components[comp] && components[comp].token === "var") {
            bindings[comp] = components[comp].value;
        } else if(components[comp] && components[comp].token === "blank") {
            bindings[comp] = "blank:"+components[comp].value;
        }
    }

    var resultsBindings =[];

    if(results!=null) {
        for(var i=0; i<results.length; i++) {
            var binding = {};
            var result  = results[i];
            for(var comp in bindings) {
                var value = result[comp];
                binding[bindings[comp]] = value;
            }
            resultsBindings.push(binding);
        }
    }

    return resultsBindings;
};


// @used
QueryPlanDPSize.areCompatibleBindings = function(bindingsa, bindingsb) {
    for(var variable in bindingsa) {
        if(bindingsb[variable]!=null && (bindingsb[variable] != bindingsa[variable])) {
            return false;
        }
    }

    return true;
};
```

```
//QueryPlanDPSize.areCompatibleBindingsStrict = function(bindingsa, bindingsb) {
//    var foundSome = false;
//    for(var variable in bindingsa) {
//  if(bindingsb[variable]!=null && (bindingsb[variable] != bindingsa[variable])) {
//      return false;
//  } else if(bindingsb[variable] == bindingsa[variable]){
//      foundSome = true;
//  }
//    }
//
//    return foundSome;
//};



    QueryPlanDPSize.mergeBindings = function(bindingsa, bindingsb) {
        var merged = {};
        for(var variable in bindingsa) {
            merged[variable] = bindingsa[variable];
        }

        for(var variable in bindingsb) {
            merged[variable] = bindingsb[variable];
        }

        return merged;
    };

    QueryPlanDPSize.joinBindings2 = function(bindingVars, bindingsa, bindingsb) {
        var acum = {};
        var bindings, variable, variableValue, values, tmp;
        var joined = [];

        for(var i=0; i<bindingsa.length; i++) {
            bindings = bindingsa[i];
            tmp = acum;
            for(var j=0; j<bindingVars.length; j++) {
                variable = bindingVars[j];
                variableValue = bindings[variable];
                if(j == bindingVars.length-1) {
                    values = tmp[variableValue] || [];
                    values.push(bindings);
                    tmp[variableValue] = values;
                } else {
                    values = tmp[variableValue] || {};
                    tmp[variableValue] = values;
                    tmp = values;
                }
            }
        }

        for(var i=0; i<bindingsb.length; i++) {
            bindings = bindingsb[i];
            tmp = acum;
            for(var j=0; j<bindingVars.length; j++) {
                variable = bindingVars[j];
                variableValue = bindings[variable];

                if(tmp[variableValue] != null) {
                    if(j == bindingVars.length-1) {
                        for(var k=0; k<tmp[variableValue].length; k++) {
                            joined.push(QueryPlanDPSize.mergeBindings(tmp[variableValue][k],bindings));
                        }
                    } else {
                        tmp = tmp[variableValue];
                    }
```

```
            }
        }
    }

    return joined;
};

QueryPlanDPSize.joinBindings = function(bindingsa, bindingsb) {
    var result = [];

    for(var i=0; i< bindingsa.length; i++) {
        var bindinga = bindingsa[i];
        for(var j=0; j<bindingsb.length; j++) {
            var bindingb = bindingsb[j];
            if(QueryPlanDPSize.areCompatibleBindings(bindinga, bindingb)){
                result.push(QueryPlanDPSize.mergeBindings(bindinga, bindingb));
            }
        }
    }
    return result;
};

QueryPlanDPSize.augmentMissingBindings = function(bindinga, bindingb) {
    for(var pb in bindingb) {
        if(bindinga[pb] == null) {
            bindinga[pb] = null;
        }
    }
    return bindinga;
};

/*
 QueryPlanDPSize.diff = function(bindingsa, biundingsb) {
 var result = [];

 for(var i=0; i< bindingsa.length; i++) {
 var bindinga = bindingsa[i];
 var matched = false;
 for(var j=0; j<bindingsb.length; j++) {
 var bindingb = bindingsb[j];
 if(QueryPlanDPSize.areCompatibleBindings(bindinga, bindingb)){
 matched = true;
 result.push(QueryPlanDPSize.mergeBindings(bindinga, bindingb));
 }
 }
 if(matched === false) {
 // missing bindings must be present for further processing
 // e.g. filtering by not present value (see DAWG tests
 // bev-6)
 QueryPlanDPSize.augmentMissingBindings(bindinga, bindingb);
 result.push(bindinga);
 }
 }

 return result;
 };
 */

QueryPlanDPSize.leftOuterJoinBindings = function(bindingsa, bindingsb) {
    var result = [];
    // strict was being passes ad an argument
    //var compatibleFunction = QueryPlanDPSize.areCompatibleBindings;
    //if(strict === true)
    //  compatibleFunction = QueryPlanDPSize.areCompatibleBindingsStrict;

    for(var i=0; i< bindingsa.length; i++) {
        var bindinga = bindingsa[i];
```

```
            var matched = false;
            for(var j=0; j<bindingsb.length; j++) {
                var bindingb = bindingsb[j];
                if(QueryPlanDPSize.areCompatibleBindings(bindinga, bindingb)){
                    matched = true;
                    result.push(QueryPlanDPSize.mergeBindings(bindinga, bindingb));
                }
            }
            if(matched === false) {
                // missing bindings must be present for further processing
                // e.g. filtering by not present value (see DAWG tests
                // bev-6)
                // augmentMissingBindings set their value to null.
                QueryPlanDPSize.augmentMissingBindings(bindinga, bindingb);
                result.push(bindinga);
            }
        }
        return result;
    };

    QueryPlanDPSize.crossProductBindings = function(bindingsa, bindingsb) {
        var result = [];

        for(var i=0; i< bindingsa.length; i++) {
            var bindinga = bindingsa[i];
            for(var j=0; j<bindingsb.length; j++) {
                var bindingb = bindingsb[j];
                result.push(QueryPlanDPSize.mergeBindings(bindinga, bindingb));
            }
        }

        return result;
    };

    QueryPlanDPSize.unionBindings = function(bindingsa, bindingsb) {
        return bindingsa.concat(bindingsb);
    };

    QueryPlanDPSize.unionManyBindings = function(bindingLists) {
        var acum = [];
        for(var i=0; i<bindingLists.length; i++) {
            var bindings = bindingLists[i];
            acum = QueryPlanDPSize.unionBindings(acum, bindings);
        }

        return acum;
    };
    return QueryPlanDPSize;
});define([
    "./query_filters",
    "../trees/utils"
], function (QueryFilters, TreeUtils) {
    var RDFJSInterface = {};

    /**
     * Implementation of <http://www.w3.org/TR/rdf-interfaces/>
     */

    // Uris map
    RDFJSInterface.defaultContext = {
        "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
        "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
        "owl": "http://www.w3.org/2002/07/owl#",
        "xsd": "http://www.w3.org/2001/XMLSchema#",
        "dcterms": "http://purl.org/dc/terms/",
        "foaf": "http://xmlns.com/foaf/0.1/",
        "cal": "http://www.w3.org/2002/12/cal/ical#",
```

```
    "vcard": "http://www.w3.org/2006/vcard/ns# ",
    "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#",
    "cc": "http://creativecommons.org/ns#",
    "sioc": "http://rdfs.org/sioc/ns#",
    "doap": "http://usefulinc.com/ns/doap#",
    "com": "http://purl.org/commerce#",
    "ps": "http://purl.org/payswarm#",
    "gr": "http://purl.org/goodrelations/v1#",
    "sig": "http://purl.org/signature#",
    "ccard": "http://purl.org/commerce/creditcard#"
};

RDFJSInterface.UrisMap = function() {
    this.defaultNs = "";
    this.interfaceProperties = ['get', 'remove', 'set', 'setDefault',
        'addAll', 'resolve', 'shrink'];
};

RDFJSInterface.UrisMap.prototype.values = function() {
    var collected = {};
    for(var p in this) {
        if(!TreeUtils.include(this.interfaceProperties,p) &&
            typeof(this[p])!=='function' &&
            p!=='defaultNs' &&
            p!=='interfaceProperties') {
            collected[p] = this[p];
        }
    }

    return collected;
};

RDFJSInterface.UrisMap.prototype.get = function(prefix) {
    if(prefix.indexOf(" ") != -1) {
        throw "Prefix must not contain any whitespaces";
    }
    return this[prefix];
};

RDFJSInterface.UrisMap.prototype.remove = function(prefix) {
    if(prefix.indexOf(" ") != -1) {
        throw "Prefix must not contain any whitespaces";
    }

    delete this[prefix];

    return null;
};

RDFJSInterface.UrisMap.prototype.set = function(prefix, iri) {
    if(prefix.indexOf(" ") != -1) {
        throw "Prefix must not contain any whitespaces";
    }

    this[prefix] = iri;
};


RDFJSInterface.UrisMap.prototype.setDefault = function(iri) {
    this.defaultNs =iri;
};

RDFJSInterface.UrisMap.prototype.addAll = function(prefixMap, override) {
    for(var prefix in prefixMap) {
        if(!TreeUtils.include(this.interfaceProperties, prefix)) {
            if(this[prefix] != null) {
                if(override === true) {
```

```
                    this[prefix] = prefixMap[prefix];
                }
            } else {
                this[prefix] = prefixMap[prefix];
            }
        }
    }

    return this;
};

RDFJSInterface.UrisMap.prototype.resolve = function(curie) {
    var parts = curie.split(":");
    var ns = parts[0];
    var suffix = parts[1];
    if(ns === '') {
        if(this.defaultNs == null) {
            return null;
        } else {
            return this.defaultNs + suffix;
        }
    } else if(this[ns] != null) {
        return this[ns] + suffix;
    } else {
        return null;
    }
};

RDFJSInterface.UrisMap.prototype.shrink = function(iri) {
    for(var ns in this) {
        var prefix = this[ns];
        if(iri.indexOf(prefix) === 0) {
            if(prefix !== '' && ns != 'defaultNs') {
                var suffix = iri.split(prefix)[1];
                return ns + ":" + suffix;
            }
        }
    }

    return iri;
};

// Profile

RDFJSInterface.Profile = function() {
    this.prefixes = new RDFJSInterface.UrisMap();
    this.terms = new RDFJSInterface.UrisMap();
};

RDFJSInterface.Profile.prototype.importProfile = function(profile, override) {
    this.prefixes.addAll(profile.prefixes, override);
    this.terms.addAll(profile.terms, override);
};


RDFJSInterface.Profile.prototype.resolve = function(toResolve) {
    if(toResolve.indexOf(":") != -1) {
        return this.prefixes.resolve(toResolve);
    } else if(this.terms[toResolve] != null) {
        return this.terms.resolve(toResolve);
    } else {
        return null;
    }
};

RDFJSInterface.Profile.prototype.setDefaultPrefix = function(iri) {
    this.prefixes.setDefault(iri);
```

```
};

RDFJSInterface.Profile.prototype.setDefaultVocabulary = function(iri) {
    this.terms.setDefault(iri);
};

RDFJSInterface.Profile.prototype.setPrefix = function(prefix, iri) {
    this.prefixes.set(prefix, iri);
};

RDFJSInterface.Profile.prototype.setTerm = function(term, iri) {
    this.terms.set(term, iri);
};

// RDF environemnt
RDFJSInterface.RDFEnvironment = function () {
    this.blankNodeCounter = 0;
    var that = this;
    this.filters = {
        s:function (s) {
            return function (t) {
                return t.subject.equals(s);
            };
        },
        p:function (p) {
            return function (t) {
                return t.predicate.equals(p);
            };
        },
        o:function (o) {
            return function (t) {
                return t.object.equals(o);
            };
        },
        sp:function (s, p) {
            return function (t) {
                return t.subject.equals(s) && t.predicate.equals(p);
            };
        },
        so:function (s, o) {
            return function (t) {
                return t.subject.equals(s) && t.object.equals(o);
            };
        },
        po:function (p, o) {
            return function (t) {
                return t.predicate.equals(p) && t.object.equals(o);
            };
        },
        spo:function (s, p, o) {
            return function (t) {
                return t.subject.equals(s) && t.predicate.equals(p) && t.object.equals(o);
            };
        },
        describes:function (v) {
            return function (t) {
                return t.subject.equals(v) || t.object.equals(v);
            };
        },
        type:function (o) {
            var type = that.resolve("rdf:type");
            return function (t) {
                return t.predicate.equals(type) && t.object.equals(o);
            };
        }
    };
```

```
    for (var p in RDFJSInterface.defaultContext) {
        this.prefixes.set(p, RDFJSInterface.defaultContext[p]);
    }
};
TreeUtils.extends(RDFJSInterface.Profile, RDFJSInterface.RDFEnvironment);

RDFJSInterface.RDFEnvironment.prototype.createBlankNode = function() {
    var bnode =  new RDFJSInterface.BlankNode(this.blankNodeCounter);
    this.blankNodeCounter++;
    return bnode;
};

RDFJSInterface.RDFEnvironment.prototype.createNamedNode = function(value) {
    var resolvedValue = this.resolve(value);
    if(resolvedValue != null) {
        return new RDFJSInterface.NamedNode(resolvedValue);
    } else {
        return new RDFJSInterface.NamedNode(value);
    }
};

RDFJSInterface.RDFEnvironment.prototype.createLiteral = function(value, language, datatype) {
    if(datatype != null) {
        return new RDFJSInterface.Literal(value, language, datatype.toString());
    } else {
        return new RDFJSInterface.Literal(value, language, datatype);
    }
};

RDFJSInterface.RDFEnvironment.prototype.createTriple = function(subject, predicate, object) {
    return new RDFJSInterface.Triple(subject, predicate, object);
};

RDFJSInterface.RDFEnvironment.prototype.createGraph = function(triples) {
    var graph = new RDFJSInterface.Graph();
    if(triples != null) {
        for(var i=0; i<triples.length; i++) {
            graph.add(triples[i]);
        }
    }
    return graph;
};

RDFJSInterface.RDFEnvironment.prototype.createAction = function(test, action) {
    return function(triple) {
        if(test(triple)) {
            return action(triple);
        } else {
            return triple;
        }
    }
};

RDFJSInterface.RDFEnvironment.prototype.createProfile = function(empty) {
    // empty (opt);
    if(empty === true) {
        return new RDFJSInterface.RDFEnvironment.Profile();
    } else {
        var profile = new RDFJSInterface.RDFEnvironment.Profile();
        profile.importProfile(this);

        return profile;
    }
};

RDFJSInterface.RDFEnvironment.prototype.createTermMap = function(empty) {
    if(empty === true) {
```

```
            return new RDFJSInterface.UrisMap();
        } else {
            var cloned = this.terms.values();
            var termMap = new RDFJSInterface.UrisMap();

            for(var p in cloned) {
                termMap[p] = cloned[p];
            }

            return termMap;
        }
    };

    RDFJSInterface.RDFEnvironment.prototype.createPrefixMap = function(empty) {
        if(empty === true) {
            return new RDFJSInterface.UrisMap();
        } else {
            var cloned = this.prefixes.values();
            var prefixMap = new RDFJSInterface.UrisMap();

            for(var p in cloned) {
                prefixMap[p] = cloned[p];
            }

            return prefixMap;
        }
    };

// Common RDFNode interface

    RDFJSInterface.RDFNode = function(interfaceName){
        this.interfaceName = interfaceName;
        this.attributes  = ["interfaceName", "nominalValue"]
    };

    RDFJSInterface.RDFNode.prototype.equals = function(otherNode) {
        if(otherNode.interfaceName == null) {
            return this.valueOf() == otherNode;

        } else {
            for(var i in this.attributes) {
                var attribute = this.attributes[i];
                if(this[attribute] != otherNode[attribute]) {
                    return false;
                }
            }

            return true;
        }
    };


// Blank node

    RDFJSInterface.BlankNode = function(bnodeId) {
        RDFJSInterface.RDFNode.call(this, "BlankNode");
        this.nominalValue = "_:"+bnodeId;
        this.bnodeId = bnodeId;
    };

    TreeUtils.extends(RDFJSInterface.RDFNode,RDFJSInterface.BlankNode);

    RDFJSInterface.BlankNode.prototype.toString = function(){
        return this.nominalValue;
    };

    RDFJSInterface.BlankNode.prototype.toNT = function() {
```

```
        return this.nominalValue;
    };

    RDFJSInterface.BlankNode.prototype.valueOf = function() {
        return this.nominalValue;
    };

// Literal node

    RDFJSInterface.Literal = function(value, language, datatype) {
        RDFJSInterface.RDFNode.call(this, "Literal");
        this.nominalValue = value;
        if(language != null) {
            this.language = language;
        } else if(datatype != null) {
            this.datatype = datatype;
        }
    };

    TreeUtils.extends(RDFJSInterface.RDFNode,RDFJSInterface.Literal);

    RDFJSInterface.Literal.prototype.toString = function(){
        var tmp = "\""+this.nominalValue+"\"";
        if(this.language != null) {
            tmp = tmp + "@" + this.language;
        } else if(this.datatype != null || this.type) {
            tmp = tmp + "^^<" + (this.datatype||this.type) + ">";
        }

        return tmp;
    };

    RDFJSInterface.Literal.prototype.toNT = function() {
        return this.toString();
    };

    RDFJSInterface.Literal.prototype.valueOf = function() {
        return QueryFilters.effectiveTypeValue({token: 'literal',
            type: (this.type || this.datatype),
            value: this.nominalValue,
            language: this.language});
    };

    // NamedNode node
    RDFJSInterface.NamedNode = function(val) {
        RDFJSInterface.RDFNode.call(this, "NamedNode");
        if(val.value != null) {
            this.nominalValue = val.value;
        } else {
            this.nominalValue = val;
        }
    };

    TreeUtils.extends(RDFJSInterface.RDFNode,RDFJSInterface.NamedNode);

    RDFJSInterface.NamedNode.prototype.toString = function(){
        return this.nominalValue;
    };

    RDFJSInterface.NamedNode.prototype.toNT = function() {
        return "<"+this.toString()+">";
    };

    RDFJSInterface.NamedNode.prototype.valueOf = function() {
        return this.nominalValue;
    };
```

```
// Triple interface
    RDFJSInterface.Triple = function(subject, predicate, object){
        this.subject = subject;
        this.predicate = predicate;
        this.object = object;
    };

    RDFJSInterface.Triple.prototype.equals = function(otherTriple) {
        return this.subject.equals(otherTriple.subject) &&
            this.predicate.equals(otherTriple.predicate) &&
            this.object.equals(otherTriple.object);
    };

    RDFJSInterface.Triple.prototype.toString = function() {
        return this.subject.toNT()+" "+this.predicate.toNT()+" "+this.object.toNT()+" . \r\n";
    };

// Graph interface

    RDFJSInterface.Graph = function() {
        this.triples = [];
        this.duplicates = {};
        this.actions = [];
        this.length = 0;
    };

    RDFJSInterface.Graph.prototype.add = function(triple) {
        for(var i=0; i<this.actions.length; i++) {
            triple = this.actions[i](triple);
        }

        var id = triple.subject.toString()+triple.predicate.toString()+triple.object.toString();
        if(!this.duplicates[id]) {
            this.duplicates[id] = true;
            this.triples.push(triple);
        }

        this.length = this.triples.length;
        return this;
    };

    RDFJSInterface.Graph.prototype.addAction = function (tripleAction, run) {
        this.actions.push(tripleAction);
        if (run == true) {
            for (var i = 0; i < this.triples.length; i++) {
                this.triples[i] = tripleAction(this.triples[i]);
            }
        }

        return this;
    };

    RDFJSInterface.Graph.prototype.addAll = function (graph) {
        var newTriples = graph.toArray();
        for (var i = 0; i < newTriples.length; i++) {
            this.add(newTriples[i]);
        }

        this.length = this.triples.length;
        return this;
    };

    RDFJSInterface.Graph.prototype.remove = function(triple) {
        var toRemove = null;
        for(var i=0; i<this.triples.length; i++) {
            if(this.triples[i].equals(triple)) {
                var id = triple.subject.toString()+triple.predicate.toString()+triple.object.toString();
```

```
            delete this.duplicates[id];
            toRemove = i;
            break;
        }
    }

    if(toRemove!=null) {
        this.triples.splice(toRemove,1);
    }

    this.length = this.triples.length;
    return this;
};

RDFJSInterface.Graph.prototype.toArray = function() {
    return this.triples;
};

RDFJSInterface.Graph.prototype.some = function(p) {
    for(var i=0; i<this.triples.length; i++) {
        if(p(this.triples[i],this) === true) {
            return true;
        }
    }

    return false;
};

RDFJSInterface.Graph.prototype.every = function(p) {
    for(var i=0; i<this.triples.length; i++) {
        if(p(this.triples[i],this) === false) {
            return false;
        }
    }

    return true;
};

RDFJSInterface.Graph.prototype.filter = function(f) {
    var tmp = new RDFJSInterface.Graph();

    for(var i=0; i<this.triples.length; i++) {
        if(f(this.triples[i],this) === true) {
            tmp.add(this.triples[i]);
        }
    }

    return tmp;
};

RDFJSInterface.Graph.prototype.forEach = function(f) {
    for(var i=0; i<this.triples.length; i++) {
        f(this.triples[i],this);
    }
};

RDFJSInterface.Graph.prototype.merge = function(g) {
    var newGraph = new RDFJSInterface.Graph();
    for(var i=0; i<this.triples.length; i++) {
        newGraph.add(this.triples[i]);
    }
    return newGraph;
};

RDFJSInterface.Graph.prototype.match = function(subject, predicate, object, limit) {
    var graph = new RDFJSInterface.Graph();
```

```
        var matched = 0;
        for(var i=0; i<this.triples.length; i++) {
            var triple = this.triples[i];
            if(subject == null || (triple.subject.equals(subject))) {
                if(predicate == null || (triple.predicate.equals(predicate))) {
                    if(object == null || (triple.object.equals(object))) {
                        if(limit==null || matched < limit) {
                            matched++;
                            graph.add(triple);
                        } else {
                            return graph;
                        }
                    }
                }
            }
        }

        return graph;
    };

    RDFJSInterface.Graph.prototype.removeMatches = function(subject, predicate, object) {
        var toRemove = [];
        for(var i=0; i<this.triples.length; i++) {
            var triple = this.triples[i];
            if(subject == null || (triple.subject.equals(subject))) {
                if(predicate == null || (triple.predicate.equals(predicate))) {
                    if(object == null || (triple.object.equals(object))) {
                        toRemove.push(triple);
                    }
                }
            }
        }

        for(var i=0; i<toRemove.length; i++) {
            this.remove(toRemove[i]);
        }

        return this;
    };

    RDFJSInterface.Graph.prototype.toNT = function() {
        var n3 = "";

        this.forEach(function(triple) {
            n3 = n3 + triple.toString();
        });

        return n3;
    };

// Builders for the query engine

    RDFJSInterface.buildRDFResource = function(value, bindings, engine, env) {
        if(value.token === 'blank') {
            return RDFJSInterface.buildBlankNode(value, bindings, engine, env);
        } else if(value.token === 'literal') {
            return RDFJSInterface.buildLiteral(value, bindings, engine, env);
        } else if(value.token === 'uri') {
            return RDFJSInterface.buildNamedNode(value, bindings, engine, env);
        } else if(value.token === 'var') {
            var result = bindings[value.value];
            if(result != null) {
                return RDFJSInterface.buildRDFResource(result, bindings, engine, env);
            } else {
                return null;
            }
        } else {
```

```
            return null;
        }
    };

    RDFJSInterface.buildBlankNode = function(value, bindings, engine, env) {
        if(value.valuetmp != null) {
            value.value = value.valuetmp;
        }
        if(value.value.indexOf("_:") === 0) {
            value.value = value.value.split("_:")[1];
        }
        return new RDFJSInterface.BlankNode(value.value);
    };

    RDFJSInterface.buildLiteral = function(value, bindings, engine, env) {
        return new RDFJSInterface.Literal(value.value, value.lang, value.type);
    };

    RDFJSInterface.buildNamedNode = function(value, bindings, engine, env) {
        if(value.value != null) {
            return new RDFJSInterface.NamedNode(value);
        } else {
            if(value.prefix != null) {
                var prefix = engine.resolveNsInEnvironment(value.prefix, env);
                value.value = prefix+value.suffix;
                return new RDFJSInterface.NamedNode(value);
            } else {
                return new RDFJSInterface.NamedNode(value);
            }
        }
    };

    RDFJSInterface.rdf = new RDFJSInterface.RDFEnvironment();
    return RDFJSInterface;
});define([], function () {
    var Lexicon = {};
    /**
     * Temporal implementation of the lexicon
     */
    Lexicon.Lexicon = function(callback){
        this.uriToOID = {};
        this.OIDToUri = {};

        this.literalToOID = {};
        this.OIDToLiteral = {};

        this.blankToOID = {};
        this.OIDToBlank = {};

        this.defaultGraphOid = 0;

        this.defaultGraphUri = "https://github.com/antoniogarrote/rdfstore-js#default_graph";
        this.defaultGraphUriTerm = {"token": "uri", "prefix": null, "suffix": null, "value":
this.defaultGraphUri, "oid": this.defaultGraphOid};
        this.oidCounter = 1;

        this.knownGraphs = {};

        if(callback != null) {
            callback(this);
        }
    };

    Lexicon.Lexicon.prototype.registerGraph = function(oid){
        if(oid != this.defaultGraphOid) {
            this.knownGraphs[oid] = true;
        }
```

```
    return true
};

Lexicon.Lexicon.prototype.registeredGraphs = function(shouldReturnUris) {
    var acum = [];

    for(var g in this.knownGraphs) {
        if(shouldReturnUris === true) {
            acum.push(this.OIDToUri['u'+g]);
        } else {
            acum.push(g);
        }
    }
    return acum;
};

Lexicon.Lexicon.prototype.registerUri = function(uri) {
    if(uri === this.defaultGraphUri) {
        return(this.defaultGraphOid);
    } else if(this.uriToOID[uri] == null){
        var oid = this.oidCounter;
        var oidStr = 'u'+oid;
        this.oidCounter++;

        this.uriToOID[uri] =[oid, 0];
        this.OIDToUri[oidStr] = uri;

        return(oid);
    } else {
        var oidCounter = this.uriToOID[uri];
        var oid = oidCounter[0];
        var counter = oidCounter[1] + 1;
        this.uriToOID[uri] = [oid, counter];
        return(oid);
    }
};

Lexicon.Lexicon.prototype.resolveUri = function(uri) {
    if(uri === this.defaultGraphUri) {
        return(this.defaultGraphOid);
    } else {
        var oidCounter = this.uriToOID[uri];
        if(oidCounter != null) {
            return(oidCounter[0]);
        } else {
            return(-1);
        }
    }
};

Lexicon.Lexicon.prototype.resolveUriCost = function(uri) {
    if(uri === this.defaultGraphUri) {
        return(this.defaultGraphOid);
    } else {
        var oidCounter = this.uriToOID[uri];
        if(oidCounter != null) {
            return(oidCounter[1]);
        } else {
            return(-1);
        }
    }
};

Lexicon.Lexicon.prototype.registerBlank = function(label) {
    var oid = this.oidCounter;
    this.oidCounter++;
    var oidStr = ""+oid;
```

```
        this.OIDToBlank[oidStr] = true;
        return(oidStr);
    };

    Lexicon.Lexicon.prototype.resolveBlank = function(label) {
        //    @todo
        //    this is failing with unicode tests... e.g. kanji2
        //    var id = label.split(":")[1];
        //    callback(id);
        var oid = this.oidCounter;
        this.oidCounter++;
        return(""+oid);
    };

    Lexicon.Lexicon.prototype.resolveBlankCost = function(label) {
        return 0;
    };

    Lexicon.Lexicon.prototype.registerLiteral = function(literal) {
        if(this.literalToOID[literal] == null){
            var oid = this.oidCounter;
            var oidStr =  'l'+ oid;
            this.oidCounter++;

            this.literalToOID[literal] = [oid, 0];
            this.OIDToLiteral[oidStr] = literal;

            return(oid);
        } else {
            var oidCounter = this.literalToOID[literal];
            var oid = oidCounter[0];
            var counter = oidCounter[1] + 1;
            this.literalToOID[literal] = [oid, counter];
            return(oid);
        }
    };

    Lexicon.Lexicon.prototype.resolveLiteral = function (literal) {
        var oidCounter = this.literalToOID[literal];
        if (oidCounter != null) {
            return(oidCounter[0]);
        } else {
            return(-1);
        }
    };

    Lexicon.Lexicon.prototype.resolveLiteralCost = function (literal) {
        var oidCounter = this.literalToOID[literal];
        if (oidCounter != null) {
            return(oidCounter[1]);
        } else {
            return(0);
        }
    };


    Lexicon.Lexicon.prototype.parseLiteral = function(literalString) {
        var parts = literalString.lastIndexOf("@");
        if(parts!=-1 && literalString[parts-1]==='"' && literalString.substring(parts,
literalString.length).match(/^@[a-zA-Z\-]+$/g)!=null) {
            var value = literalString.substring(1,parts-1);
            var lang = literalString.substring(parts+1, literalString.length);
            return {token: "literal", value:value, lang:lang};
        }

        var parts = literalString.lastIndexOf("^^");
        if(parts!=-1 && literalString[parts-1]==='"' && literalString[parts+2] === '<' && literalString
```

```
[literalString.length-1] === '>') {
            var value = literalString.substring(1,parts-1);
            var type = literalString.substring(parts+3, literalString.length-1);

            return {token: "literal", value:value, type:type};
        }

        var value = literalString.substring(1,literalString.length-1);
        return {token:"literal", value:value};
    };

    Lexicon.Lexicon.prototype.parseUri = function(uriString) {
        return {token: "uri", value:uriString};
    };

    Lexicon.Lexicon.prototype.retrieve = function(oid) {
        try {
            if(oid === this.defaultGraphOid) {
                return({ token: "uri",
                    value:this.defaultGraphUri,
                    prefix: null,
                    suffix: null,
                    defaultGraph: true });
            } else {
                var maybeUri = this.OIDToUri['u'+oid];
                if(maybeUri != null) {
                    return(this.parseUri(maybeUri));
                } else {
                    var maybeLiteral = this.OIDToLiteral['l'+oid];
                    if(maybeLiteral != null) {
                        return(this.parseLiteral(maybeLiteral));
                    } else {
                        var maybeBlank = this.OIDToBlank[""+oid];
                        if(maybeBlank != null) {
                            return({token:"blank", value:"_:"+oid});
                        } else {
                            throw("Null value for OID");
                        }
                    }
                }
            }
        } catch(e) {
            //console.log("error in lexicon retrieving OID:");
            //console.log(oid);
            if(e.message || e.stack) {
                if(e.message) {
                    //console.log(e.message);
                }
                if(e.stack) {
                    //console.log(e.stack);
                }
            } else {
                //console.log(e);
            }
            throw new Error("Unknown retrieving OID in lexicon:"+oid);

        }
    };

    Lexicon.Lexicon.prototype.clear = function() {
        this.uriToOID = {};
        this.OIDToUri = {};

        this.literalToOID = {};
        this.OIDToLiteral = {};

        this.blankToOID = {};
```

```
        this.OIDToBlank = {};
    };

    Lexicon.Lexicon.prototype.unregister = function (quad, key) {
        try {
            this.unregisterTerm(quad.subject.token, key.subject);
            this.unregisterTerm(quad.predicate.token, key.predicate);
            this.unregisterTerm(quad.object.token, key.object);
            if (quad.graph != null) {
                this.unregisterTerm(quad.graph.token, key.graph);
            }
            return(true);
        } catch (e) {
            //console.log("Error unregistering quad");
            //console.log(e.message);
            return(false);
        }
    };

    Lexicon.Lexicon.prototype.unregisterTerm = function (kind, oid) {
        if (kind === 'uri') {
            if (oid != this.defaultGraphOid) {
                var oidStr = 'u' + oid;
                var uri = this.OIDToUri[oidStr];     // = uri;
                var oidCounter = this.uriToOID[uri]; // =[oid, 0];

                var counter = oidCounter[1];
                if ("" + oidCounter[0] === "" + oid) {
                    if (counter === 0) {
                        delete this.OIDToUri[oidStr];
                        delete this.uriToOID[uri];
                        // delete the graph oid from known graphs
                        // in case this URI is a graph identifier
                        delete this.knownGraphs[oid];
                    } else {
                        this.uriToOID[uri] = [oid, counter - 1];
                    }
                } else {
                    throw("Not matching OID : " + oid + " vs " + oidCounter[0]);
                }
            }
        } else if (kind === 'literal') {
            this.oidCounter++;
            var oidStr = 'l' + oid;
            var literal = this.OIDToLiteral[oidStr];  // = literal;
            var oidCounter = this.literalToOID[literal]; // = [oid, 0];

            var counter = oidCounter[1];
            if ("" + oidCounter[0] === "" + oid) {
                if (counter === 0) {
                    delete this.OIDToLiteral[oidStr];
                    delete this.literalToOID[literal];
                } else {
                    this.literalToOID[literal] = [oid, counter - 1];
                }
            } else {
                throw("Not matching OID : " + oid + " vs " + oidCounter[0]);
            }

        } else if (kind === 'blank') {
            delete this.OIDToBlank["" + oid];
        }
    };
    return Lexicon;
});

define(["../trees/utils", "./quad_index"], function (TreeUtils, QuadIndex) {
```

```
var QuadBackend = {};
/*
 * "perfect" indices for RDF indexing
 *
 * SPOG (?, ?, ?, ?), (s, ?, ?, ?), (s, p, ?, ?), (s, p, o, ?), (s, p, o, g)
 * GP   (?, ?, ?, g), (?, p, ?, g)
 * OGS  (?, ?, o, ?), (?, ?, o, g), (s, ?, o, g)
 * POG  (?, p, ?, ?), (?, p, o, ?), (?, p, o, g)
 * GSP  (s, ?, ?, g), (s, p, ?, g)
 * OS   (s, ?, o, ?)
 */
QuadBackend.QuadBackend = function (configuration, callback) {
    if (arguments != 0) {
        this.indexMap = {};
        this.treeOrder = configuration['treeOrder'];
        this.indices = ['SPOG', 'GP', 'OGS', 'POG', 'GSP', 'OS'];
        this.componentOrders = {
            SPOG:['subject', 'predicate', 'object', 'graph'],
            GP:['graph', 'predicate', 'subject', 'object'],
            OGS:['object', 'graph', 'subject', 'predicate'],
            POG:['predicate', 'object', 'graph', 'subject'],
            GSP:['graph', 'subject', 'predicate', 'object'],
            OS:['object', 'subject', 'predicate', 'graph']
        };

        for (var i = 0; i < this.indices.length; i++) {
            var indexKey = this.indices[i];
            this.indexMap[indexKey] = new QuadIndex.Tree({order:this.treeOrder,
                componentOrder:this.componentOrders[indexKey],
                persistent:configuration['persistent'],
                name:(configuration['name'] || "") + indexKey,
                cacheMaxSize:configuration['cacheMaxSize']});
        }

        if (callback)
            callback(this);
    }
};

QuadBackend.QuadBackend.prototype.clear = function() {
    for(var i=0; i<this.indices.length; i++) {
        var indexKey = this.indices[i];
        this.indexMap[indexKey].clear();
    }
};

QuadBackend.QuadBackend.prototype._indexForPattern = function (pattern) {
    var indexKey = pattern.indexKey;
    var matchingIndices = this.indices;

    for (var i = 0; i < matchingIndices.length; i++) {
        var index = matchingIndices[i];
        var indexComponents = this.componentOrders[index];
        for (var j = 0; j < indexComponents.length; j++) {
            if (TreeUtils.include(indexKey, indexComponents[j]) === false) {
                break;
            }
            if (j == indexKey.length - 1) {
                return index;
            }
        }
    }

    return 'SPOG'; // If no other match, we erturn the more generic index
};
```

```
    QuadBackend.QuadBackend.prototype.index = function (quad, callback) {
        for (var i = 0; i < this.indices.length; i++) {
            var indexKey = this.indices[i];
            var index = this.indexMap[indexKey];

            index.insert(quad);
        }

        if (callback)
            callback(true);

        return true;
    };

    QuadBackend.QuadBackend.prototype.range = function (pattern, callback) {
        var indexKey = this._indexForPattern(pattern);
        var index = this.indexMap[indexKey];
        var quads = index.range(pattern);
        if (callback)
            callback(quads);

        return quads;
    };

    QuadBackend.QuadBackend.prototype.search = function (quad, callback) {
        var indexKey = this.indices[0];
        var index = this.indexMap[indexKey];
        var result = index.search(quad);

        if (callback)
            callback(result != null);

        return (result != null)
    };


    QuadBackend.QuadBackend.prototype.delete = function (quad, callback) {
        var indexKey, index;
        for (var i = 0; i < this.indices.length; i++) {
            indexKey = this.indices[i];
            index = this.indexMap[indexKey];

            index.delete(quad);
        }

        if (callback)
            callback(true);

        return true;
    };
    return QuadBackend;
});define([], function () {
    var QuadIndexCommon = {};
    /**
     * NodeKey
     *
     * Implements the interface of BinarySearchTree.Node
     *
     * A Tree node augmented with BPlusTree
     * node structures
     *
     * @param components
     * @param [order]
     * @constructor
     */
    QuadIndexCommon.NodeKey = function(components, order) {
        this.subject = components.subject;
```

```
        this.predicate = components.predicate;
        this.object = components.object;
        this.graph = components.graph;
        this.order = order;
    };

    QuadIndexCommon.NodeKey.prototype.comparator = function(keyPattern) {
        for(var i=0; i<this.order.length; i++) {
            var component = this.order[i];
            if(keyPattern[component] == null) {
                return 0;
            } else {
                if(this[component] < keyPattern[component] ) {
                    return -1
                } else if(this[component] > keyPattern[component]) {
                    return 1
                }
            }
        }

        return 0;
    };

    /**
     * Pattern
     *
     * A pattern with some variable components
     */
    QuadIndexCommon.Pattern = function (components) {
        this.subject = components.subject;
        this.predicate = components.predicate;
        this.object = components.object;
        this.graph = components.graph;
        this.indexKey = [];

        this.keyComponents = {};

        var order = [];
        var indif = [];
        components = ['subject', 'predicate', 'object', 'graph'];

        // components must have been already normalized and
        // inserted in the lexicon.
        // OIDs retrieved from the lexicon *are* numbers so
        // they can be told apart from variables (strings)
        for (var i = 0; i < components.length; i++) {
            if (typeof(this[components[i]]) === 'string') {
                indif.push(components[i]);
                this.keyComponents[components[i]] = null;
            } else {
                order.push(components[i]);
                this.keyComponents[components[i]] = this[components[i]];
                this.indexKey.push(components[i]);
            }
        }

        this.order = order.concat(indif);
        this.key = new QuadIndexCommon.NodeKey(this.keyComponents, this.order);
    };
    return QuadIndexCommon;
});

define(["../trees/in_memory_b_tree", "../trees/utils"], function (BaseTree, Utils) {
    var QuadIndex = {};

    QuadIndex.Tree = function(params,callback) {
        if(arguments != 0) {
```

```
            this.componentOrder = params.componentOrder;


            // @todo change this if using the file backed implementation
            BaseTree.Tree.call(this, params.order, params['name'], params['persistent'], params
['cacheMaxSize']);

            this.comparator = function (a, b) {
                for (var i = 0; i < this.componentOrder.length; i++) {
                    var component = this.componentOrder[i];
                    var vala = a[component];
                    var valb = b[component];
                    if (vala < valb) {
                        return -1;
                    } else if (vala > valb) {
                        return 1;
                    }
                }
                return 0;
            };

            this.rangeComparator = function (a, b) {
                for (var i = 0; i < this.componentOrder.length; i++) {
                    var component = this.componentOrder[i];
                    if (b[component] == null || a[component] == null) {
                        return 0;
                    } else {
                        if (a[component] < b[component]) {
                            return -1
                        } else if (a[component] > b[component]) {
                            return 1
                        }
                    }
                }

                return 0;
            };

            if(callback!=null) {
                callback(this);
            }
        }
    };

    Utils.extends(BaseTree.Tree, QuadIndex.Tree);

    QuadIndex.Tree.prototype.insert = function(quad, callback) {
        BaseTree.Tree.prototype.insert.call(this, quad, null);
        if(callback)
            callback(true);

        return true
    };

    QuadIndex.Tree.prototype.search = function(quad, callback) {
        var result = BaseTree.Tree.prototype.search.call(this, quad, true); // true -> check exists : not
present in all the b-tree implementations, check first.
        if(callback)
            callback(result);

        return result;
    };

    QuadIndex.Tree.prototype.range = function (pattern, callback) {
        var result = null;
        if (typeof(this.root) === 'string') {
            result = this._rangeTraverse(this, this._diskRead(this.root), pattern);
```

```
        } else {
            result = this._rangeTraverse(this, this.root, pattern);
        }

        if (callback)
            callback(result);

        return result;
    };

    QuadIndex.Tree.prototype._rangeTraverse = function(tree,node, pattern) {
        var patternKey  = pattern.key;
        var acum = [];
        var pendingNodes = [node];
        var node, idxMin, idxMax;
        while(pendingNodes.length > 0) {
            node = pendingNodes.shift();
            idxMin = 0;

            while(idxMin < node.numberActives && tree.rangeComparator(node.keys[idxMin].key,patternKey)
=== -1) {
                idxMin++;
            }
            if(node.isLeaf === true) {
                idxMax = idxMin;

                while(idxMax < node.numberActives && tree.rangeComparator(node.keys
[idxMax].key,patternKey) === 0) {
                    acum.push(node.keys[idxMax].key);
                    idxMax++;
                }

            } else {
                var pointer = node.children[idxMin];
                var childNode = tree._diskRead(pointer);
                pendingNodes.push(childNode);

                var idxMax = idxMin;
                while(true) {
                    if(idxMax < node.numberActives && tree.rangeComparator(node.keys
[idxMax].key,patternKey) === 0) {
                        acum.push(node.keys[idxMax].key);
                        idxMax++;
                        childNode = tree._diskRead(node.children[idxMax]);
                        pendingNodes.push(childNode);
                    } else {
                        break;
                    }
                }
            }
        }
        return acum;
    };
    return QuadIndex;
})/*global define */
define([], function () {
    var SparqlParser = {};

    SparqlParser.parser = (function(){
        /* Generated by PEG.js 0.6.2 (http://pegjs.majda.cz/). */

        var result = {
            /*
             * Parses the input with a generated parser. If the parsing is successfull,
             * returns a value explicitly or implicitly specified by the grammar from
             * which the parser was generated (see |PEG.buildParser|). If the parsing is
             * unsuccessful, throws |PEG.parser.SyntaxError| describing the error.
```

```
     */
    parse: function(input, startRule) {
        var parseFunctions = {
            "ANON": parse_ANON,
            "AdditiveExpression": parse_AdditiveExpression,
            "Aggregate": parse_Aggregate,
            "ArgList": parse_ArgList,
            "AskQuery": parse_AskQuery,
            "BLANK_NODE_LABEL": parse_BLANK_NODE_LABEL,
            "BaseDecl": parse_BaseDecl,
            "BindingValue": parse_BindingValue,
            "BindingsClause": parse_BindingsClause,
            "BlankNode": parse_BlankNode,
            "BlankNodePropertyList": parse_BlankNodePropertyList,
            "BooleanLiteral": parse_BooleanLiteral,
            "BrackettedExpression": parse_BrackettedExpression,
            "BuiltInCall": parse_BuiltInCall,
            "COMMENT": parse_COMMENT,
            "Clear": parse_Clear,
            "Collection": parse_Collection,
            "ConditionalAndExpression": parse_ConditionalAndExpression,
            "ConditionalOrExpression": parse_ConditionalOrExpression,
            "Constraint": parse_Constraint,
            "ConstructQuery": parse_ConstructQuery,
            "ConstructTemplate": parse_ConstructTemplate,
            "ConstructTriples": parse_ConstructTriples,
            "Create": parse_Create,
            "DECIMAL": parse_DECIMAL,
            "DECIMAL_NEGATIVE": parse_DECIMAL_NEGATIVE,
            "DECIMAL_POSITIVE": parse_DECIMAL_POSITIVE,
            "DOUBLE": parse_DOUBLE,
            "DOUBLE_NEGATIVE": parse_DOUBLE_NEGATIVE,
            "DOUBLE_POSITIVE": parse_DOUBLE_POSITIVE,
            "DatasetClause": parse_DatasetClause,
            "DefaultGraphClause": parse_DefaultGraphClause,
            "DeleteClause": parse_DeleteClause,
            "DeleteData": parse_DeleteData,
            "DeleteWhere": parse_DeleteWhere,
            "DescribeQuery": parse_DescribeQuery,
            "Drop": parse_Drop,
            "ECHAR": parse_ECHAR,
            "EXPONENT": parse_EXPONENT,
            "ExistsFunc": parse_ExistsFunc,
            "ExpressionList": parse_ExpressionList,
            "Filter": parse_Filter,
            "FunctionCall": parse_FunctionCall,
            "GraphGraphPattern": parse_GraphGraphPattern,
            "GraphNode": parse_GraphNode,
            "GraphPatternNotTriples": parse_GraphPatternNotTriples,
            "GraphRef": parse_GraphRef,
            "GraphRefAll": parse_GraphRefAll,
            "GraphTerm": parse_GraphTerm,
            "GroupClause": parse_GroupClause,
            "GroupCondition": parse_GroupCondition,
            "GroupGraphPattern": parse_GroupGraphPattern,
            "GroupGraphPatternSub": parse_GroupGraphPatternSub,
            "GroupOrUnionGraphPattern": parse_GroupOrUnionGraphPattern,
            "HavingClause": parse_HavingClause,
            "INTEGER": parse_INTEGER,
            "INTEGER_NEGATIVE": parse_INTEGER_NEGATIVE,
            "INTEGER_POSITIVE": parse_INTEGER_POSITIVE,
            "IRI_REF": parse_IRI_REF,
            "IRIref": parse_IRIref,
            "IRIrefOrFunction": parse_IRIrefOrFunction,
            "InsertClause": parse_InsertClause,
            "InsertData": parse_InsertData,
            "LANGTAG": parse_LANGTAG,
```

```
"LimitClause": parse_LimitClause,
"LimitOffsetClauses": parse_LimitOffsetClauses,
"Load": parse_Load,
"MinusGraphPattern": parse_MinusGraphPattern,
"Modify": parse_Modify,
"MultiplicativeExpression": parse_MultiplicativeExpression,
"NIL": parse_NIL,
"NamedGraphClause": parse_NamedGraphClause,
"NotExistsFunc": parse_NotExistsFunc,
"NumericLiteral": parse_NumericLiteral,
"NumericLiteralNegative": parse_NumericLiteralNegative,
"NumericLiteralPositive": parse_NumericLiteralPositive,
"NumericLiteralUnsigned": parse_NumericLiteralUnsigned,
"ObjectList": parse_ObjectList,
"OffsetClause": parse_OffsetClause,
"OptionalGraphPattern": parse_OptionalGraphPattern,
"OrderClause": parse_OrderClause,
"OrderCondition": parse_OrderCondition,
"PNAME_LN": parse_PNAME_LN,
"PNAME_NS": parse_PNAME_NS,
"PN_CHARS": parse_PN_CHARS,
"PN_CHARS_BASE": parse_PN_CHARS_BASE,
"PN_CHARS_U": parse_PN_CHARS_U,
"PN_LOCAL": parse_PN_LOCAL,
"PN_PREFIX": parse_PN_PREFIX,
"PathAlternative": parse_PathAlternative,
"PathElt": parse_PathElt,
"PathEltOrInverse": parse_PathEltOrInverse,
"PathMod": parse_PathMod,
"PathNegatedPropertySet": parse_PathNegatedPropertySet,
"PathOneInPropertySet": parse_PathOneInPropertySet,
"PathPrimary": parse_PathPrimary,
"PathSequence": parse_PathSequence,
"PrefixDecl": parse_PrefixDecl,
"PrefixedName": parse_PrefixedName,
"PrimaryExpression": parse_PrimaryExpression,
"Prologue": parse_Prologue,
"PropertyList": parse_PropertyList,
"PropertyListNotEmpty": parse_PropertyListNotEmpty,
"PropertyListNotEmptyPath": parse_PropertyListNotEmptyPath,
"PropertyListPath": parse_PropertyListPath,
"QuadData": parse_QuadData,
"QuadPattern": parse_QuadPattern,
"Quads": parse_Quads,
"QuadsNotTriples": parse_QuadsNotTriples,
"Query": parse_Query,
"RDFLiteral": parse_RDFLiteral,
"RegexExpression": parse_RegexExpression,
"RelationalExpression": parse_RelationalExpression,
"SPARQL": parse_SPARQL,
"STRING_LITERAL1": parse_STRING_LITERAL1,
"STRING_LITERAL2": parse_STRING_LITERAL2,
"STRING_LITERAL_LONG1": parse_STRING_LITERAL_LONG1,
"STRING_LITERAL_LONG2": parse_STRING_LITERAL_LONG2,
"SelectClause": parse_SelectClause,
"SelectQuery": parse_SelectQuery,
"ServiceGraphPattern": parse_ServiceGraphPattern,
"SolutionModifier": parse_SolutionModifier,
"String": parse_String,
"SubSelect": parse_SubSelect,
"TriplesBlock": parse_TriplesBlock,
"TriplesNode": parse_TriplesNode,
"TriplesSameSubject": parse_TriplesSameSubject,
"TriplesSameSubjectPath": parse_TriplesSameSubjectPath,
"TriplesTemplate": parse_TriplesTemplate,
"UnaryExpression": parse_UnaryExpression,
"Update": parse_Update,
```

```
        "Update1": parse_Update1,
        "UsingClause": parse_UsingClause,
        "VAR1": parse_VAR1,
        "VAR2": parse_VAR2,
        "VARNAME": parse_VARNAME,
        "Var": parse_Var,
        "VarOrIRIref": parse_VarOrIRIref,
        "VarOrTerm": parse_VarOrTerm,
        "Verb": parse_Verb,
        "VerbPath": parse_VerbPath,
        "WS": parse_WS,
        "WhereClause": parse_WhereClause
    };

    if (startRule !== undefined) {
        if (parseFunctions[startRule] === undefined) {
            throw new Error("Invalid rule name: " + quote(startRule) + ".");
        }
    } else {
        startRule = "SPARQL";
    }

    var pos = 0;
    var reportMatchFailures = true;
    var rightmostMatchFailuresPos = 0;
    var rightmostMatchFailuresExpected = [];
    var cache = {};

    function padLeft(input, padding, length) {
        var result = input;

        var padLength = length - input.length;
        for (var i = 0; i < padLength; i++) {
            result = padding + result;
        }

        return result;
    }

    function escape(ch) {
        var charCode = ch.charCodeAt(0);

        if (charCode <= 0xFF) {
            var escapeChar = 'x';
            var length = 2;
        } else {
            var escapeChar = 'u';
            var length = 4;
        }

        return '\\' + escapeChar + padLeft(charCode.toString(16).toUpperCase(), '0', length);
    }

    function quote(s) {
        /*
         * ECMA-262, 5th ed., 7.8.4: All characters may appear literally in a
         * string literal except for the closing quote character, backslash,
         * carriage return, line separator, paragraph separator, and line feed.
         * Any character may appear in the form of an escape sequence.
         */
        return '"' + s
            .replace(/\\/g, '\\\\')            // backslash
            .replace(/"/g, '\\"')              // closing quote character
            .replace(/\r/g, '\\r')             // carriage return
            .replace(/\n/g, '\\n')             // line feed
            .replace(/[\x80-\uFFFF]/g, escape) // non-ASCII characters
            + '"';
```

```
}

function matchFailed(failure) {
    if (pos < rightmostMatchFailuresPos) {
        return;
    }

    if (pos > rightmostMatchFailuresPos) {
        rightmostMatchFailuresPos = pos;
        rightmostMatchFailuresExpected = [];
    }

    rightmostMatchFailuresExpected.push(failure);
}

function parse_SPARQL() {
    var cacheKey = 'SPARQL@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }


    var result2 = parse_Query();
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result1 = parse_Update();
        if (result1 !== null) {
            var result0 = result1;
        } else {
            var result0 = null;;
        };
    }



    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_Query() {
    var cacheKey = 'Query@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    var savedPos1 = pos;
    var result3 = parse_Prologue();
    if (result3 !== null) {
        var result8 = parse_SelectQuery();
        if (result8 !== null) {
            var result4 = result8;
        } else {
            var result7 = parse_ConstructQuery();
            if (result7 !== null) {
                var result4 = result7;
            } else {
```

```
                    var result6 = parse_DescribeQuery();
                    if (result6 !== null) {
                        var result4 = result6;
                    } else {
                        var result5 = parse_AskQuery();
                        if (result5 !== null) {
                            var result4 = result5;
                        } else {
                            var result4 = null;;
                        };
                    };
                };
            }
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(p, q) {
            return {token: 'query',
                kind: 'query',
                prologue: p,
                units: [q]};
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[2] Query");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Prologue() {
        var cacheKey = 'Prologue@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result8 = parse_BaseDecl();
        var result3 = result8 !== null ? result8 : '';
        if (result3 !== null) {
            var result4 = [];
            var result7 = parse_WS();
            while (result7 !== null) {
```

```
                result4.push(result7);
                var result7 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = [];
                var result6 = parse_PrefixDecl();
                while (result6 !== null) {
                    result5.push(result6);
                    var result6 = parse_PrefixDecl();
                }
                if (result5 !== null) {
                    var result1 = [result3, result4, result5];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(b, pfx) {
            return { token: 'prologue',
                base: b,
                prefixes: pfx }
        })(result1[0], result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[3] Prologue");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_BaseDecl() {
        var cacheKey = 'BaseDecl@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result10 = parse_WS();
        while (result10 !== null) {
            result3.push(result10);
            var result10 = parse_WS();
        }
```

```
    if (result3 !== null) {
        if (input.substr(pos, 4) === "BASE") {
            var result9 = "BASE";
            pos += 4;
        } else {
            var result9 = null;
            if (reportMatchFailures) {
                matchFailed("\"BASE\"");
            }
        }
        if (result9 !== null) {
            var result4 = result9;
        } else {
            if (input.substr(pos, 4) === "base") {
                var result8 = "base";
                pos += 4;
            } else {
                var result8 = null;
                if (reportMatchFailures) {
                    matchFailed("\"base\"");
                }
            }
            if (result8 !== null) {
                var result4 = result8;
            } else {
                var result4 = null;;
            };
        }
        if (result4 !== null) {
            var result5 = [];
            var result7 = parse_WS();
            while (result7 !== null) {
                result5.push(result7);
                var result7 = parse_WS();
            }
            if (result5 !== null) {
                var result6 = parse_IRI_REF();
                if (result6 !== null) {
                    var result1 = [result3, result4, result5, result6];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(i) {
        registerDefaultPrefix(i);

        var base = {};
        base.token = 'base';
        base.value = i;

        return base;
    })(result1[3])
        : null;
    if (result2 !== null) {
```

```
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[4] BaseDecl");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PrefixDecl() {
        var cacheKey = 'PrefixDecl@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result13 = parse_WS();
        while (result13 !== null) {
            result3.push(result13);
            var result13 = parse_WS();
        }
        if (result3 !== null) {
            if (input.substr(pos, 6) === "PREFIX") {
                var result12 = "PREFIX";
                pos += 6;
            } else {
                var result12 = null;
                if (reportMatchFailures) {
                    matchFailed("\"PREFIX\"");
                }
            }
            if (result12 !== null) {
                var result4 = result12;
            } else {
                if (input.substr(pos, 6) === "prefix") {
                    var result11 = "prefix";
                    pos += 6;
                } else {
                    var result11 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"prefix\"");
                    }
                }
                if (result11 !== null) {
                    var result4 = result11;
                } else {
                    var result4 = null;;
                };
            }
            if (result4 !== null) {
                var result5 = [];
                var result10 = parse_WS();
                while (result10 !== null) {
```

```
                            result5.push(result10);
                            var result10 = parse_WS();
                        }
                        if (result5 !== null) {
                            var result6 = parse_PNAME_NS();
                            if (result6 !== null) {
                                var result7 = [];
                                var result9 = parse_WS();
                                while (result9 !== null) {
                                    result7.push(result9);
                                    var result9 = parse_WS();
                                }
                                if (result7 !== null) {
                                    var result8 = parse_IRI_REF();
                                    if (result8 !== null) {
                                        var result1 = [result3, result4, result5, result6, result7,
result8];
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function(p, l) {

                    registerPrefix(p,l);

                    var prefix = {};
                    prefix.token = 'prefix';
                    prefix.prefix = p;
                    prefix.local = l;

                    return prefix;
                })(result1[3], result1[5])
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[5] PrefixDecl");
                }

                cache[cacheKey] = {
                    nextPos: pos,
```

```
                    result:  result0
                };
                return result0;
            }

            function parse_SelectQuery() {
                var cacheKey = 'SelectQuery@' + pos;
                var cachedResult = cache[cacheKey];
                if (cachedResult) {
                    pos = cachedResult.nextPos;
                    return cachedResult.result;
                }

                var savedReportMatchFailures = reportMatchFailures;
                reportMatchFailures = false;
                var savedPos0 = pos;
                var savedPos1 = pos;
                var result3 = parse_SelectClause();
                if (result3 !== null) {
                    var result4 = [];
                    var result16 = parse_WS();
                    while (result16 !== null) {
                        result4.push(result16);
                        var result16 = parse_WS();
                    }
                    if (result4 !== null) {
                        var result5 = [];
                        var result15 = parse_DatasetClause();
                        while (result15 !== null) {
                            result5.push(result15);
                            var result15 = parse_DatasetClause();
                        }
                        if (result5 !== null) {
                            var result6 = [];
                            var result14 = parse_WS();
                            while (result14 !== null) {
                                result6.push(result14);
                                var result14 = parse_WS();
                            }
                            if (result6 !== null) {
                                var result7 = parse_WhereClause();
                                if (result7 !== null) {
                                    var result8 = [];
                                    var result13 = parse_WS();
                                    while (result13 !== null) {
                                        result8.push(result13);
                                        var result13 = parse_WS();
                                    }
                                    if (result8 !== null) {
                                        var result9 = parse_SolutionModifier();
                                        if (result9 !== null) {
                                            var result10 = [];
                                            var result12 = parse_WS();
                                            while (result12 !== null) {
                                                result10.push(result12);
                                                var result12 = parse_WS();
                                            }
                                            if (result10 !== null) {
                                                var result11 = parse_BindingsClause();
                                                if (result11 !== null) {
                                                    var result1 = [result3, result4, result5,
result6, result7, result8, result9, result10, result11];
                                                } else {
                                                    var result1 = null;
                                                    pos = savedPos1;
                                                }
                                            } else {
```

```
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(s, gs, w, sm) {

        var dataset = {'named':[], 'implicit':[]};
        for(var i=0; i<gs.length; i++) {
            var g = gs[i];
            if(g.kind === 'default') {
                dataset['implicit'].push(g.graph);
            } else {
                dataset['named'].push(g.graph)
            }
        }


        if(dataset['named'].length === 0 && dataset['implicit'].length === 0) {
            dataset['implicit'].push({token:'uri',
                prefix:null,
                suffix:null,
                value:'https://github.com/antoniogarrote/rdfstore-js#default_graph'});
        }

        var query = {};
        query.kind = 'select';
        query.token = 'executableunit'
        query.dataset = dataset;
        query.projection = s.vars;
        query.modifier = s.modifier;
        query.pattern = w

        if(sm!=null && sm.limit!=null) {
            query.limit = sm.limit;
        }
        if(sm!=null && sm.offset!=null) {
            query.offset = sm.offset;
        }
        if(sm!=null && (sm.order!=null && sm.order!="")) {
```

```
                query.order = sm.order;
            }
            if(sm!=null && sm.group!=null) {
                query.group = sm.group;
            }

            return query
        })(result1[0], result1[2], result1[4], result1[6])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[6] SelectQuery");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_SubSelect() {
        var cacheKey = 'SubSelect@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var result1 = parse_SelectClause();
        if (result1 !== null) {
            var result2 = parse_WhereClause();
            if (result2 !== null) {
                var result3 = parse_SolutionModifier();
                if (result3 !== null) {
                    var result0 = [result1, result2, result3];
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
            } else {
                var result0 = null;
                pos = savedPos0;
            }
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[7] SubSelect");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
```

```
    }

    function parse_SelectClause() {
        var cacheKey = 'SelectClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result54 = parse_WS();
        while (result54 !== null) {
            result3.push(result54);
            var result54 = parse_WS();
        }
        if (result3 !== null) {
            if (input.substr(pos, 6) === "SELECT") {
                var result53 = "SELECT";
                pos += 6;
            } else {
                var result53 = null;
                if (reportMatchFailures) {
                    matchFailed("\"SELECT\"");
                }
            }
            if (result53 !== null) {
                var result4 = result53;
            } else {
                if (input.substr(pos, 6) === "select") {
                    var result52 = "select";
                    pos += 6;
                } else {
                    var result52 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"select\"");
                    }
                }
                if (result52 !== null) {
                    var result4 = result52;
                } else {
                    var result4 = null;;
                };
            }
            if (result4 !== null) {
                var result5 = [];
                var result51 = parse_WS();
                while (result51 !== null) {
                    result5.push(result51);
                    var result51 = parse_WS();
                }
                if (result5 !== null) {
                    if (input.substr(pos, 8) === "DISTINCT") {
                        var result50 = "DISTINCT";
                        pos += 8;
                    } else {
                        var result50 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"DISTINCT\"");
                        }
                    }
                    if (result50 !== null) {
                        var result48 = result50;
```

```
} else {
    if (input.substr(pos, 8) === "distinct") {
        var result49 = "distinct";
        pos += 8;
    } else {
        var result49 = null;
        if (reportMatchFailures) {
            matchFailed("\"distinct\"");
        }
    }
    if (result49 !== null) {
        var result48 = result49;
    } else {
        var result48 = null;;
    };
}
if (result48 !== null) {
    var result44 = result48;
} else {
    if (input.substr(pos, 7) === "REDUCED") {
        var result47 = "REDUCED";
        pos += 7;
    } else {
        var result47 = null;
        if (reportMatchFailures) {
            matchFailed("\"REDUCED\"");
        }
    }
    if (result47 !== null) {
        var result45 = result47;
    } else {
        if (input.substr(pos, 7) === "reduced") {
            var result46 = "reduced";
            pos += 7;
        } else {
            var result46 = null;
            if (reportMatchFailures) {
                matchFailed("\"reduced\"");
            }
        }
        if (result46 !== null) {
            var result45 = result46;
        } else {
            var result45 = null;;
        };
    }
    if (result45 !== null) {
        var result44 = result45;
    } else {
        var result44 = null;;
    };
}
var result6 = result44 !== null ? result44 : '';
if (result6 !== null) {
    var result7 = [];
    var result43 = parse_WS();
    while (result43 !== null) {
        result7.push(result43);
        var result43 = parse_WS();
    }
    if (result7 !== null) {
        var savedPos4 = pos;
        var result38 = [];
        var result42 = parse_WS();
        while (result42 !== null) {
            result38.push(result42);
            var result42 = parse_WS();
```

```
                }
                if (result38 !== null) {
                    var result39 = parse_Var();
                    if (result39 !== null) {
                        var result40 = [];
                        var result41 = parse_WS();
                        while (result41 !== null) {
                            result40.push(result41);
                            var result41 = parse_WS();
                        }
                        if (result40 !== null) {
                            var result37 = [result38, result39, result40];
                        } else {
                            var result37 = null;
                            pos = savedPos4;
                        }
                    } else {
                        var result37 = null;
                        pos = savedPos4;
                    }
                } else {
                    var result37 = null;
                    pos = savedPos4;
                }
                if (result37 !== null) {
                    var result16 = result37;
                } else {
                    var savedPos3 = pos;
                    var result18 = [];
                    var result36 = parse_WS();
                    while (result36 !== null) {
                        result18.push(result36);
                        var result36 = parse_WS();
                    }
                    if (result18 !== null) {
                        if (input.substr(pos, 1) === "(") {
                            var result19 = "(";
                            pos += 1;
                        } else {
                            var result19 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"(\"");
                            }
                        }
                        if (result19 !== null) {
                            var result20 = [];
                            var result35 = parse_WS();
                            while (result35 !== null) {
                                result20.push(result35);
                                var result35 = parse_WS();
                            }
                            if (result20 !== null) {
                                var result21 = parse_ConditionalOrExpression();
                                if (result21 !== null) {
                                    var result22 = [];
                                    var result34 = parse_WS();
                                    while (result34 !== null) {
                                        result22.push(result34);
                                        var result34 = parse_WS();
                                    }
                                    if (result22 !== null) {
                                        if (input.substr(pos, 2) === "AS") {
                                            var result33 = "AS";
                                            pos += 2;
                                        } else {
                                            var result33 = null;
                                            if (reportMatchFailures) {
```

```
                                                                        matchFailed("\"AS\"");
                                                                    }
                                                                }
                                                                if (result33 !== null) {
                                                                    var result23 = result33;
                                                                } else {
                                                                    if (input.substr(pos, 2) === "as") {
                                                                        var result32 = "as";
                                                                        pos += 2;
                                                                    } else {
                                                                        var result32 = null;
                                                                        if (reportMatchFailures) {
                                                                            matchFailed("\"as\"");
                                                                        }
                                                                    }
                                                                    if (result32 !== null) {
                                                                        var result23 = result32;
                                                                    } else {
                                                                        var result23 = null;;
                                                                    };
                                                                }
                                                                if (result23 !== null) {
                                                                    var result24 = [];
                                                                    var result31 = parse_WS();
                                                                    while (result31 !== null) {
                                                                        result24.push(result31);
                                                                        var result31 = parse_WS();
                                                                    }
                                                                    if (result24 !== null) {
                                                                        var result25 = parse_Var();
                                                                        if (result25 !== null) {
                                                                            var result26 = [];
                                                                            var result30 = parse_WS();
                                                                            while (result30 !== null) {
                                                                                result26.push(result30);
                                                                                var result30 = parse_WS();
                                                                            }
                                                                            if (result26 !== null) {
                                                                                if (input.substr(pos, 1)
=== ")") {

                                                                                    var result27 = ")";
                                                                                    pos += 1;
                                                                                } else {
                                                                                    var result27 = null;
                                                                                    if
(reportMatchFailures) {

                                                                                        matchFailed("\")
\"");

                                                                                    }
                                                                                }
                                                                                if (result27 !== null) {
                                                                                    var result28 = [];
                                                                                    var result29 =
parse_WS();

                                                                                    while (result29 !==
null) {

                                                                                        result28.push
(result29);

                                                                                        var result29 =
parse_WS();

                                                                                    }
                                                                                    if (result28 !==
null) {

                                                                                        var result17 =
[result18, result19, result20, result21, result22, result23, result24, result25, result26, result27,
result28];

                                                                                    } else {
```

```
                                                                    var result17 =
null;
                                                                    pos = savedPos3;
                                                                }
                                                            } else {
                                                                var result17 = null;
                                                                pos = savedPos3;
                                                            }
                                                        } else {
                                                            var result17 = null;
                                                            pos = savedPos3;
                                                        }
                                                    } else {
                                                        var result17 = null;
                                                        pos = savedPos3;
                                                    }
                                                } else {
                                                    var result17 = null;
                                                    pos = savedPos3;
                                                }
                                            } else {
                                                var result17 = null;
                                                pos = savedPos3;
                                            }
                                        } else {
                                            var result17 = null;
                                            pos = savedPos3;
                                        }
                                    } else {
                                        var result17 = null;
                                        pos = savedPos3;
                                    }
                                } else {
                                    var result17 = null;
                                    pos = savedPos3;
                                }
                            } else {
                                var result17 = null;
                                pos = savedPos3;
                            }
                        } else {
                            var result17 = null;
                            pos = savedPos3;
                        }
                        if (result17 !== null) {
                            var result16 = result17;
                        } else {
                            var result16 = null;;
                        };
                    }
                    if (result16 !== null) {
                        var result15 = [];
                        while (result16 !== null) {
                            result15.push(result16);
                            var savedPos4 = pos;
                            var result38 = [];
                            var result42 = parse_WS();
                            while (result42 !== null) {
                                result38.push(result42);
                                var result42 = parse_WS();
                            }
                            if (result38 !== null) {
                                var result39 = parse_Var();
                                if (result39 !== null) {
                                    var result40 = [];
                                    var result41 = parse_WS();
                                    while (result41 !== null) {
```

```
                                    result40.push(result41);
                                    var result41 = parse_WS();
                                }
                                if (result40 !== null) {
                                    var result37 = [result38, result39, result40];
                                } else {
                                    var result37 = null;
                                    pos = savedPos4;
                                }
                            } else {
                                var result37 = null;
                                pos = savedPos4;
                            }
                        } else {
                            var result37 = null;
                            pos = savedPos4;
                        }
                        if (result37 !== null) {
                            var result16 = result37;
                        } else {
                            var savedPos3 = pos;
                            var result18 = [];
                            var result36 = parse_WS();
                            while (result36 !== null) {
                                result18.push(result36);
                                var result36 = parse_WS();
                            }
                            if (result18 !== null) {
                                if (input.substr(pos, 1) === "(") {
                                    var result19 = "(";
                                    pos += 1;
                                } else {
                                    var result19 = null;
                                    if (reportMatchFailures) {
                                        matchFailed("\"(\"");
                                    }
                                }
                                if (result19 !== null) {
                                    var result20 = [];
                                    var result35 = parse_WS();
                                    while (result35 !== null) {
                                        result20.push(result35);
                                        var result35 = parse_WS();
                                    }
                                    if (result20 !== null) {
                                        var result21 =
parse_ConditionalOrExpression();
                                        if (result21 !== null) {
                                            var result22 = [];
                                            var result34 = parse_WS();
                                            while (result34 !== null) {
                                                result22.push(result34);
                                                var result34 = parse_WS();
                                            }
                                            if (result22 !== null) {
                                                if (input.substr(pos, 2) ===
"AS") {
                                                    var result33 = "AS";
                                                    pos += 2;
                                                } else {
                                                    var result33 = null;
                                                    if (reportMatchFailures) {
                                                        matchFailed("\"AS\"");
                                                    }
                                                }
                                                if (result33 !== null) {
                                                    var result23 = result33;
```

```
"as") {
                                                                    } else {
                                                                        if (input.substr(pos, 2) ===

                                                                            var result32 = "as";
                                                                            pos += 2;
                                                                        } else {
                                                                            var result32 = null;
                                                                            if (reportMatchFailures) {
                                                                                matchFailed("\"as\"");
                                                                            }
                                                                        }
                                                                        if (result32 !== null) {
                                                                            var result23 = result32;
                                                                        } else {
                                                                            var result23 = null;;
                                                                        };
                                                                    }
                                                                    if (result23 !== null) {
                                                                        var result24 = [];
                                                                        var result31 = parse_WS();
                                                                        while (result31 !== null) {
                                                                            result24.push(result31);
                                                                            var result31 = parse_WS();
                                                                        }
                                                                        if (result24 !== null) {
                                                                            var result25 = parse_Var
();


parse_WS();

null) {

(result30);

parse_WS();


null) {

(pos, 1) === ")") {

= ")";


= null;

(reportMatchFailures) {

matchFailed("\")\"");


null) {

= [];

= parse_WS();

(result29 !== null) {

result28.push(result29);

result29 = parse_WS();
                                                                                if (result25 !== null) {
                                                                                    var result26 = [];
                                                                                    var result30 =

                                                                                    while (result30 !==

                                                                                        result26.push

                                                                                        var result30 =

                                                                                    }
                                                                                    if (result26 !==

                                                                                        if (input.substr

                                                                                            var result27

                                                                                            pos += 1;
                                                                                        } else {
                                                                                            var result27

                                                                                            if



                                                                                        }
                                                                                    }
                                                                                    if (result27 !==

                                                                                        var result28

                                                                                        var result29

                                                                                        while


                                                                                            var

                                                                                        }
```

```
                                                                                    if (result28 !
== null) {
                                                                                        var
result17 = [result18, result19, result20, result21, result22, result23, result24, result25, result26,
result27, result28];
                                                                                    } else {
                                                                                        var
result17 = null;
                                                                                        pos =
savedPos3;
                                                                                    }
                                                                                } else {
                                                                                    var result17
= null;
                                                                                    pos =
savedPos3;
                                                                                }
                                                                            } else {
                                                                                var result17 =
null;
                                                                                pos = savedPos3;
                                                                            }
                                                                        } else {
                                                                            var result17 = null;
                                                                            pos = savedPos3;
                                                                        }
                                                                    } else {
                                                                        var result17 = null;
                                                                        pos = savedPos3;
                                                                    }
                                                                } else {
                                                                    var result17 = null;
                                                                    pos = savedPos3;
                                                                }
                                                            } else {
                                                                var result17 = null;
                                                                pos = savedPos3;
                                                            }
                                                        } else {
                                                            var result17 = null;
                                                            pos = savedPos3;
                                                        }
                                                    } else {
                                                        var result17 = null;
                                                        pos = savedPos3;
                                                    }
                                                } else {
                                                    var result17 = null;
                                                    pos = savedPos3;
                                                }
                                            } else {
                                                var result17 = null;
                                                pos = savedPos3;
                                            }
                                            if (result17 !== null) {
                                                var result16 = result17;
                                            } else {
                                                var result16 = null;;
                                            };
                                        }
                                    }
                                } else {
                                    var result15 = null;
                                }
                                if (result15 !== null) {
                                    var result8 = result15;
                                } else {
```

```
                                    var savedPos2 = pos;
                                    var result10 = [];
                                    var result14 = parse_WS();
                                    while (result14 !== null) {
                                        result10.push(result14);
                                        var result14 = parse_WS();
                                    }
                                    if (result10 !== null) {
                                        if (input.substr(pos, 1) === "*") {
                                            var result11 = "*";
                                            pos += 1;
                                        } else {
                                            var result11 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\"*\"");
                                            }
                                        }
                                        if (result11 !== null) {
                                            var result12 = [];
                                            var result13 = parse_WS();
                                            while (result13 !== null) {
                                                result12.push(result13);
                                                var result13 = parse_WS();
                                            }
                                            if (result12 !== null) {
                                                var result9 = [result10, result11, result12];
                                            } else {
                                                var result9 = null;
                                                pos = savedPos2;
                                            }
                                        } else {
                                            var result9 = null;
                                            pos = savedPos2;
                                        }
                                    } else {
                                        var result9 = null;
                                        pos = savedPos2;
                                    }
                                    if (result9 !== null) {
                                        var result8 = result9;
                                    } else {
                                        var result8 = null;;
                                    };
                                }
                                if (result8 !== null) {
                                    var result1 = [result3, result4, result5, result6, result7,
result8];
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
```

```
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                    var result2 = result1 !== null
                        ? (function(mod, proj) {
                        var vars = [];
                        if(proj.length === 3 && proj[1]==="*") {
                            return {vars: [{token: 'variable', kind:'*'}], modifier:arrayToString(mod)};
                        }

                        for(var i=0; i< proj.length; i++) {
                            var aVar = proj[i];

                            if(aVar.length === 3) {
                                vars.push({token: 'variable', kind:'var', value:aVar[1]});
                            } else {
                                vars.push({token: 'variable', kind:'aliased', expression: aVar[3],
alias:aVar[7]})
                            }
                        }

                        return {vars: vars, modifier:arrayToString(mod)};
                    })(result1[3], result1[5])
                        : null;
                    if (result2 !== null) {
                        var result0 = result2;
                    } else {
                        var result0 = null;
                        pos = savedPos0;
                    }
                    reportMatchFailures = savedReportMatchFailures;
                    if (reportMatchFailures && result0 === null) {
                        matchFailed("[8] SelectClause");
                    }

                    cache[cacheKey] = {
                        nextPos: pos,
                        result:  result0
                    };
                    return result0;
                }

                function parse_ConstructQuery() {
                    var cacheKey = 'ConstructQuery@' + pos;
                    var cachedResult = cache[cacheKey];
                    if (cachedResult) {
                        pos = cachedResult.nextPos;
                        return cachedResult.result;
                    }

                    var savedReportMatchFailures = reportMatchFailures;
                    reportMatchFailures = false;
                    var savedPos0 = pos;
                    var savedPos1 = pos;
                    var result3 = [];
                    var result20 = parse_WS();
                    while (result20 !== null) {
                        result3.push(result20);
                        var result20 = parse_WS();
                    }
                    if (result3 !== null) {
                        if (input.substr(pos, 9) === "CONSTRUCT") {
                            var result19 = "CONSTRUCT";
                            pos += 9;
                        } else {
                            var result19 = null;
```

```
                    if (reportMatchFailures) {
                        matchFailed("\"CONSTRUCT\"");
                    }
                }
                if (result19 !== null) {
                    var result4 = result19;
                } else {
                    if (input.substr(pos, 9) === "construct") {
                        var result18 = "construct";
                        pos += 9;
                    } else {
                        var result18 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"construct\"");
                        }
                    }
                    if (result18 !== null) {
                        var result4 = result18;
                    } else {
                        var result4 = null;;
                    };
                }
                if (result4 !== null) {
                    var result5 = [];
                    var result17 = parse_WS();
                    while (result17 !== null) {
                        result5.push(result17);
                        var result17 = parse_WS();
                    }
                    if (result5 !== null) {
                        var result6 = parse_ConstructTemplate();
                        if (result6 !== null) {
                            var result7 = [];
                            var result16 = parse_WS();
                            while (result16 !== null) {
                                result7.push(result16);
                                var result16 = parse_WS();
                            }
                            if (result7 !== null) {
                                var result8 = [];
                                var result15 = parse_DatasetClause();
                                while (result15 !== null) {
                                    result8.push(result15);
                                    var result15 = parse_DatasetClause();
                                }
                                if (result8 !== null) {
                                    var result9 = [];
                                    var result14 = parse_WS();
                                    while (result14 !== null) {
                                        result9.push(result14);
                                        var result14 = parse_WS();
                                    }
                                    if (result9 !== null) {
                                        var result10 = parse_WhereClause();
                                        if (result10 !== null) {
                                            var result11 = [];
                                            var result13 = parse_WS();
                                            while (result13 !== null) {
                                                result11.push(result13);
                                                var result13 = parse_WS();
                                            }
                                            if (result11 !== null) {
                                                var result12 = parse_SolutionModifier();
                                                if (result12 !== null) {
                                                    var result1 = [result3, result4, result5,
result6, result7, result8, result9, result10, result11, result12];
                                                } else {
```

```
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(t, gs, w, sm) {
        var dataset = {'named':[], 'implicit':[]};
        for(var i=0; i<gs.length; i++) {
            var g = gs[i];
            if(g.kind === 'default') {
                dataset['implicit'].push(g.graph);
            } else {
                dataset['named'].push(g.graph)
            }
        }


        if(dataset['named'].length === 0 && dataset['implicit'].length === 0) {
            dataset['implicit'].push({token:'uri',
                prefix:null,
                suffix:null,
                value:'https://github.com/antoniogarrote/rdfstore-js#default_graph'});
        }

        var query = {};
        query.kind = 'construct';
        query.token = 'executableunit'
        query.dataset = dataset;
        query.template = t;
        query.pattern = w

        if(sm!=null && sm.limit!=null) {
```

```
                query.limit = sm.limit;
            }
            if(sm!=null && sm.offset!=null) {
                query.offset = sm.offset;
            }
            if(sm!=null && (sm.order!=null && sm.order!="")) {
                query.order = sm.order;
            }
            return query

        })(result1[3], result1[5], result1[7], result1[9])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[9] ConstructQuery");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DescribeQuery() {
        var cacheKey = 'DescribeQuery@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        if (input.substr(pos, 8) === "DESCRIBE") {
            var result1 = "DESCRIBE";
            pos += 8;
        } else {
            var result1 = null;
            if (reportMatchFailures) {
                matchFailed("\"DESCRIBE\"");
            }
        }
        if (result1 !== null) {
            var result10 = parse_VarOrIRIref();
            if (result10 !== null) {
                var result9 = [];
                while (result10 !== null) {
                    result9.push(result10);
                    var result10 = parse_VarOrIRIref();
                }
            } else {
                var result9 = null;
            }
            if (result9 !== null) {
                var result2 = result9;
            } else {
                if (input.substr(pos, 1) === "*") {
                    var result8 = "*";
                    pos += 1;
```

```
                } else {
                    var result8 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"*\"");
                    }
                }
                if (result8 !== null) {
                    var result2 = result8;
                } else {
                    var result2 = null;;
                };
            }
            if (result2 !== null) {
                var result3 = [];
                var result7 = parse_DatasetClause();
                while (result7 !== null) {
                    result3.push(result7);
                    var result7 = parse_DatasetClause();
                }
                if (result3 !== null) {
                    var result6 = parse_WhereClause();
                    var result4 = result6 !== null ? result6 : '';
                    if (result4 !== null) {
                        var result5 = parse_SolutionModifier();
                        if (result5 !== null) {
                            var result0 = [result1, result2, result3, result4, result5];
                        } else {
                            var result0 = null;
                            pos = savedPos0;
                        }
                    } else {
                        var result0 = null;
                        pos = savedPos0;
                    }
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
            } else {
                var result0 = null;
                pos = savedPos0;
            }
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[10] DescribeQuery");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_AskQuery() {
        var cacheKey = 'AskQuery@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
```

```
                    reportMatchFailures = false;
                    var savedPos0 = pos;
                    var savedPos1 = pos;
                    var result3 = [];
                    var result14 = parse_WS();
                    while (result14 !== null) {
                        result3.push(result14);
                        var result14 = parse_WS();
                    }
                    if (result3 !== null) {
                        if (input.substr(pos, 3) === "ASK") {
                            var result13 = "ASK";
                            pos += 3;
                        } else {
                            var result13 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"ASK\"");
                            }
                        }
                        if (result13 !== null) {
                            var result4 = result13;
                        } else {
                            if (input.substr(pos, 3) === "ask") {
                                var result12 = "ask";
                                pos += 3;
                            } else {
                                var result12 = null;
                                if (reportMatchFailures) {
                                    matchFailed("\"ask\"");
                                }
                            }
                            if (result12 !== null) {
                                var result4 = result12;
                            } else {
                                var result4 = null;;
                            };
                        }
                        if (result4 !== null) {
                            var result5 = [];
                            var result11 = parse_WS();
                            while (result11 !== null) {
                                result5.push(result11);
                                var result11 = parse_WS();
                            }
                            if (result5 !== null) {
                                var result6 = [];
                                var result10 = parse_DatasetClause();
                                while (result10 !== null) {
                                    result6.push(result10);
                                    var result10 = parse_DatasetClause();
                                }
                                if (result6 !== null) {
                                    var result7 = [];
                                    var result9 = parse_WS();
                                    while (result9 !== null) {
                                        result7.push(result9);
                                        var result9 = parse_WS();
                                    }
                                    if (result7 !== null) {
                                        var result8 = parse_WhereClause();
                                        if (result8 !== null) {
                                            var result1 = [result3, result4, result5, result6, result7,
result8];
                                        } else {
                                            var result1 = null;
                                            pos = savedPos1;
                                        }
```

```
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(gs, w) {
        var dataset = {'named':[], 'implicit':[]};
        for(var i=0; i<gs.length; i++) {
            var g = gs[i];
            if(g.kind === 'implicit') {
                dataset['implicit'].push(g.graph);
            } else {
                dataset['named'].push(g.graph)
            }
        }


        if(dataset['named'].length === 0 && dataset['implicit'].length === 0) {
            dataset['implicit'].push({token:'uri',
                prefix:null,
                suffix:null,
                value:'https://github.com/antoniogarrote/rdfstore-js#default_graph'});
        }

        var query = {};
        query.kind = 'ask';
        query.token = 'executableunit'
        query.dataset = dataset;
        query.pattern = w

        return query
    })(result1[3], result1[5])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[11] AskQuery");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}
```

```
        function parse_DatasetClause() {
            var cacheKey = 'DatasetClause@' + pos;
            var cachedResult = cache[cacheKey];
            if (cachedResult) {
                pos = cachedResult.nextPos;
                return cachedResult.result;
            }

            var savedReportMatchFailures = reportMatchFailures;
            reportMatchFailures = false;
            var savedPos0 = pos;
            var savedPos1 = pos;
            if (input.substr(pos, 4) === "FROM") {
                var result12 = "FROM";
                pos += 4;
            } else {
                var result12 = null;
                if (reportMatchFailures) {
                    matchFailed("\"FROM\"");
                }
            }
            if (result12 !== null) {
                var result3 = result12;
            } else {
                if (input.substr(pos, 4) === "from") {
                    var result11 = "from";
                    pos += 4;
                } else {
                    var result11 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"from\"");
                    }
                }
                if (result11 !== null) {
                    var result3 = result11;
                } else {
                    var result3 = null;;
                };
            }
            if (result3 !== null) {
                var result4 = [];
                var result10 = parse_WS();
                while (result10 !== null) {
                    result4.push(result10);
                    var result10 = parse_WS();
                }
                if (result4 !== null) {
                    var result9 = parse_DefaultGraphClause();
                    if (result9 !== null) {
                        var result5 = result9;
                    } else {
                        var result8 = parse_NamedGraphClause();
                        if (result8 !== null) {
                            var result5 = result8;
                        } else {
                            var result5 = null;;
                        };
                    }
                    if (result5 !== null) {
                        var result6 = [];
                        var result7 = parse_WS();
                        while (result7 !== null) {
                            result6.push(result7);
                            var result7 = parse_WS();
                        }
                        if (result6 !== null) {
                            var result1 = [result3, result4, result5, result6];
```

```
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(gs) {
            return gs;
        })(result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[12] DatasetClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DefaultGraphClause() {
        var cacheKey = 'DefaultGraphClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result5 = parse_WS();
        while (result5 !== null) {
            result3.push(result5);
            var result5 = parse_WS();
        }
        if (result3 !== null) {
            var result4 = parse_IRIref();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
```

```
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(s) {
            return {graph:s , kind:'default', token:'graphClause'}
        })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[13] DefaultGraphClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_NamedGraphClause() {
        var cacheKey = 'NamedGraphClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 5) === "NAMED") {
            var result8 = "NAMED";
            pos += 5;
        } else {
            var result8 = null;
            if (reportMatchFailures) {
                matchFailed("\"NAMED\"");
            }
        }
        if (result8 !== null) {
            var result3 = result8;
        } else {
            if (input.substr(pos, 5) === "named") {
                var result7 = "named";
                pos += 5;
            } else {
                var result7 = null;
                if (reportMatchFailures) {
                    matchFailed("\"named\"");
                }
            }
            if (result7 !== null) {
                var result3 = result7;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result6 = parse_WS();
```

```
            while (result6 !== null) {
                result4.push(result6);
                var result6 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = parse_IRIref();
                if (result5 !== null) {
                    var result1 = [result3, result4, result5];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(s) {
            return {graph:s, kind:'named', token:'graphCluase'};
        })(result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[14] NamedGraphClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_WhereClause() {
        var cacheKey = 'WhereClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 5) === "WHERE") {
            var result11 = "WHERE";
            pos += 5;
        } else {
            var result11 = null;
            if (reportMatchFailures) {
                matchFailed("\"WHERE\"");
            }
        }
        if (result11 !== null) {
            var result9 = result11;
        } else {
```

```
        if (input.substr(pos, 5) === "where") {
            var result10 = "where";
            pos += 5;
        } else {
            var result10 = null;
            if (reportMatchFailures) {
                matchFailed("\"where\"");
            }
        }
        if (result10 !== null) {
            var result9 = result10;
        } else {
            var result9 = null;;
        };
    }
    var result3 = result9 !== null ? result9 : '';
    if (result3 !== null) {
        var result4 = [];
        var result8 = parse_WS();
        while (result8 !== null) {
            result4.push(result8);
            var result8 = parse_WS();
        }
        if (result4 !== null) {
            var result5 = parse_GroupGraphPattern();
            if (result5 !== null) {
                var result6 = [];
                var result7 = parse_WS();
                while (result7 !== null) {
                    result6.push(result7);
                    var result7 = parse_WS();
                }
                if (result6 !== null) {
                    var result1 = [result3, result4, result5, result6];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(g) {
        return g;
    })(result1[2])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[16] WhereClause");
    }

    cache[cacheKey] = {
```

```
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_SolutionModifier() {
        var cacheKey = 'SolutionModifier@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result10 = parse_GroupClause();
        var result3 = result10 !== null ? result10 : '';
        if (result3 !== null) {
            var result9 = parse_HavingClause();
            var result4 = result9 !== null ? result9 : '';
            if (result4 !== null) {
                var result8 = parse_OrderClause();
                var result5 = result8 !== null ? result8 : '';
                if (result5 !== null) {
                    var result7 = parse_LimitOffsetClauses();
                    var result6 = result7 !== null ? result7 : '';
                    if (result6 !== null) {
                        var result1 = [result3, result4, result5, result6];
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(gc, oc, lo) {
            var acum = {};
            if(lo != null) {
                if(lo.limit != null) {
                    acum.limit = lo.limit;
                }
                if(lo.offset != null) {
                    acum.offset = lo.offset;
                }
            }

            if(gc != null) {
                acum.group = gc;
            }

            acum.order = oc;

            return acum
        })(result1[0], result1[2], result1[3])
```

```
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[17] SolutionModifier");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_GroupClause() {
        var cacheKey = 'GroupClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 5) === "GROUP") {
            var result14 = "GROUP";
            pos += 5;
        } else {
            var result14 = null;
            if (reportMatchFailures) {
                matchFailed("\"GROUP\"");
            }
        }
        if (result14 !== null) {
            var result3 = result14;
        } else {
            if (input.substr(pos, 5) === "group") {
                var result13 = "group";
                pos += 5;
            } else {
                var result13 = null;
                if (reportMatchFailures) {
                    matchFailed("\"group\"");
                }
            }
            if (result13 !== null) {
                var result3 = result13;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result12 = parse_WS();
            while (result12 !== null) {
                result4.push(result12);
                var result12 = parse_WS();
            }
            if (result4 !== null) {
                if (input.substr(pos, 2) === "BY") {
```

```
                var result11 = "BY";
                pos += 2;
            } else {
                var result11 = null;
                if (reportMatchFailures) {
                    matchFailed("\"BY\"");
                }
            }
            if (result11 !== null) {
                var result5 = result11;
            } else {
                if (input.substr(pos, 2) === "by") {
                    var result10 = "by";
                    pos += 2;
                } else {
                    var result10 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"by\"");
                    }
                }
                if (result10 !== null) {
                    var result5 = result10;
                } else {
                    var result5 = null;;
                };
            }
            if (result5 !== null) {
                var result6 = [];
                var result9 = parse_WS();
                while (result9 !== null) {
                    result6.push(result9);
                    var result9 = parse_WS();
                }
                if (result6 !== null) {
                    var result8 = parse_GroupCondition();
                    if (result8 !== null) {
                        var result7 = [];
                        while (result8 !== null) {
                            result7.push(result8);
                            var result8 = parse_GroupCondition();
                        }
                    } else {
                        var result7 = null;
                    }
                    if (result7 !== null) {
                        var result1 = [result3, result4, result5, result6, result7];
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
```

```
            ? (function(conds) {
            return conds;
        })(result1[4])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[18] GroupClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_GroupCondition() {
        var cacheKey = 'GroupCondition@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos7 = pos;
        var savedPos8 = pos;
        var result44 = [];
        var result48 = parse_WS();
        while (result48 !== null) {
            result44.push(result48);
            var result48 = parse_WS();
        }
        if (result44 !== null) {
            var result45 = parse_BuiltInCall();
            if (result45 !== null) {
                var result46 = [];
                var result47 = parse_WS();
                while (result47 !== null) {
                    result46.push(result47);
                    var result47 = parse_WS();
                }
                if (result46 !== null) {
                    var result42 = [result44, result45, result46];
                } else {
                    var result42 = null;
                    pos = savedPos8;
                }
            } else {
                var result42 = null;
                pos = savedPos8;
            }
        } else {
            var result42 = null;
            pos = savedPos8;
        }
        var result43 = result42 !== null
            ? (function(b) {
            return b;
        })(result42[1])
```

```
    : null;
if (result43 !== null) {
    var result41 = result43;
} else {
    var result41 = null;
    pos = savedPos7;
}
if (result41 !== null) {
    var result0 = result41;
} else {
    var savedPos5 = pos;
    var savedPos6 = pos;
    var result36 = [];
    var result40 = parse_WS();
    while (result40 !== null) {
        result36.push(result40);
        var result40 = parse_WS();
    }
    if (result36 !== null) {
        var result37 = parse_FunctionCall();
        if (result37 !== null) {
            var result38 = [];
            var result39 = parse_WS();
            while (result39 !== null) {
                result38.push(result39);
                var result39 = parse_WS();
            }
            if (result38 !== null) {
                var result34 = [result36, result37, result38];
            } else {
                var result34 = null;
                pos = savedPos6;
            }
        } else {
            var result34 = null;
            pos = savedPos6;
        }
    } else {
        var result34 = null;
        pos = savedPos6;
    }
    var result35 = result34 !== null
        ? (function(f) {
        return f;
    })(result34[1])
        : null;
    if (result35 !== null) {
        var result33 = result35;
    } else {
        var result33 = null;
        pos = savedPos5;
    }
    if (result33 !== null) {
        var result0 = result33;
    } else {
        var savedPos2 = pos;
        var savedPos3 = pos;
        var result12 = [];
        var result32 = parse_WS();
        while (result32 !== null) {
            result12.push(result32);
            var result32 = parse_WS();
        }
        if (result12 !== null) {
            if (input.substr(pos, 1) === "(") {
                var result13 = "(";
                pos += 1;
```

```
} else {
    var result13 = null;
    if (reportMatchFailures) {
        matchFailed("\"(\"");
    }
}
if (result13 !== null) {
    var result14 = [];
    var result31 = parse_WS();
    while (result31 !== null) {
        result14.push(result31);
        var result31 = parse_WS();
    }
    if (result14 !== null) {
        var result15 = parse_ConditionalOrExpression();
        if (result15 !== null) {
            var result16 = [];
            var result30 = parse_WS();
            while (result30 !== null) {
                result16.push(result30);
                var result30 = parse_WS();
            }
            if (result16 !== null) {
                var savedPos4 = pos;
                if (input.substr(pos, 2) === "AS") {
                    var result29 = "AS";
                    pos += 2;
                } else {
                    var result29 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"AS\"");
                    }
                }
                if (result29 !== null) {
                    var result24 = result29;
                } else {
                    if (input.substr(pos, 2) === "as") {
                        var result28 = "as";
                        pos += 2;
                    } else {
                        var result28 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"as\"");
                        }
                    }
                    if (result28 !== null) {
                        var result24 = result28;
                    } else {
                        var result24 = null;;
                    };
                }
                if (result24 !== null) {
                    var result25 = [];
                    var result27 = parse_WS();
                    while (result27 !== null) {
                        result25.push(result27);
                        var result27 = parse_WS();
                    }
                    if (result25 !== null) {
                        var result26 = parse_Var();
                        if (result26 !== null) {
                            var result23 = [result24, result25, result26];
                        } else {
                            var result23 = null;
                            pos = savedPos4;
                        }
                    } else {
```

```
                                                    var result23 = null;
                                                    pos = savedPos4;
                                                }
                                            } else {
                                                var result23 = null;
                                                pos = savedPos4;
                                            }
                                            var result17 = result23 !== null ? result23 : '';
                                            if (result17 !== null) {
                                                var result18 = [];
                                                var result22 = parse_WS();
                                                while (result22 !== null) {
                                                    result18.push(result22);
                                                    var result22 = parse_WS();
                                                }
                                                if (result18 !== null) {
                                                    if (input.substr(pos, 1) === ")") {
                                                        var result19 = ")";
                                                        pos += 1;
                                                    } else {
                                                        var result19 = null;
                                                        if (reportMatchFailures) {
                                                            matchFailed("\")\"");
                                                        }
                                                    }
                                                    if (result19 !== null) {
                                                        var result20 = [];
                                                        var result21 = parse_WS();
                                                        while (result21 !== null) {
                                                            result20.push(result21);
                                                            var result21 = parse_WS();
                                                        }
                                                        if (result20 !== null) {
                                                            var result10 = [result12, result13,
result14, result15, result16, result17, result18, result19, result20];
                                                        } else {
                                                            var result10 = null;
                                                            pos = savedPos3;
                                                        }
                                                    } else {
                                                        var result10 = null;
                                                        pos = savedPos3;
                                                    }
                                                } else {
                                                    var result10 = null;
                                                    pos = savedPos3;
                                                }
                                            } else {
                                                var result10 = null;
                                                pos = savedPos3;
                                            }
                                        } else {
                                            var result10 = null;
                                            pos = savedPos3;
                                        }
                                    } else {
                                        var result10 = null;
                                        pos = savedPos3;
                                    }
                                } else {
                                    var result10 = null;
                                    pos = savedPos3;
                                }
                            } else {
                                var result10 = null;
                                pos = savedPos3;
                            }
```

```
} else {
    var result10 = null;
    pos = savedPos3;
}
var result11 = result10 !== null
    ? (function(e, alias) {
    if(alias.length != 0) {
        return {token: 'aliased_expression',
            expression: e,
            alias: alias[2] };
    } else {
        return e;
    }
})(result10[3], result10[5])
    : null;
if (result11 !== null) {
    var result9 = result11;
} else {
    var result9 = null;
    pos = savedPos2;
}
if (result9 !== null) {
    var result0 = result9;
} else {
    var savedPos0 = pos;
    var savedPos1 = pos;
    var result4 = [];
    var result8 = parse_WS();
    while (result8 !== null) {
        result4.push(result8);
        var result8 = parse_WS();
    }
    if (result4 !== null) {
        var result5 = parse_Var();
        if (result5 !== null) {
            var result6 = [];
            var result7 = parse_WS();
            while (result7 !== null) {
                result6.push(result7);
                var result7 = parse_WS();
            }
            if (result6 !== null) {
                var result2 = [result4, result5, result6];
            } else {
                var result2 = null;
                pos = savedPos1;
            }
        } else {
            var result2 = null;
            pos = savedPos1;
        }
    } else {
        var result2 = null;
        pos = savedPos1;
    }
    var result3 = result2 !== null
        ? (function(v) {
        return v;
    })(result2[1])
        : null;
    if (result3 !== null) {
        var result1 = result3;
    } else {
        var result1 = null;
        pos = savedPos0;
    }
    if (result1 !== null) {
```

```
                    var result0 = result1;
                } else {
                    var result0 = null;;
                };
            };
        };
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[19] GroupCondition");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_HavingClause() {
    var cacheKey = 'HavingClause@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    if (input.substr(pos, 6) === "HAVING") {
        var result1 = "HAVING";
        pos += 6;
    } else {
        var result1 = null;
        if (reportMatchFailures) {
            matchFailed("\"HAVING\"");
        }
    }
    if (result1 !== null) {
        var result3 = parse_Constraint();
        if (result3 !== null) {
            var result2 = [];
            while (result3 !== null) {
                result2.push(result3);
                var result3 = parse_Constraint();
            }
        } else {
            var result2 = null;
        }
        if (result2 !== null) {
            var result0 = [result1, result2];
        } else {
            var result0 = null;
            pos = savedPos0;
        }
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[20] HavingClause");
    }

    cache[cacheKey] = {
        nextPos: pos,
```

```
            result:  result0
        };
        return result0;
    }

    function parse_OrderClause() {
        var cacheKey = 'OrderClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 5) === "ORDER") {
            var result16 = "ORDER";
            pos += 5;
        } else {
            var result16 = null;
            if (reportMatchFailures) {
                matchFailed("\"ORDER\"");
            }
        }
        if (result16 !== null) {
            var result3 = result16;
        } else {
            if (input.substr(pos, 5) === "order") {
                var result15 = "order";
                pos += 5;
            } else {
                var result15 = null;
                if (reportMatchFailures) {
                    matchFailed("\"order\"");
                }
            }
            if (result15 !== null) {
                var result3 = result15;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result14 = parse_WS();
            while (result14 !== null) {
                result4.push(result14);
                var result14 = parse_WS();
            }
            if (result4 !== null) {
                if (input.substr(pos, 2) === "BY") {
                    var result13 = "BY";
                    pos += 2;
                } else {
                    var result13 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"BY\"");
                    }
                }
                if (result13 !== null) {
                    var result5 = result13;
                } else {
                    if (input.substr(pos, 2) === "by") {
                        var result12 = "by";
                        pos += 2;
```

```
                                } else {
                                    var result12 = null;
                                    if (reportMatchFailures) {
                                        matchFailed("\"by\"");
                                    }
                                }
                                if (result12 !== null) {
                                    var result5 = result12;
                                } else {
                                    var result5 = null;;
                                };
                            }
                            if (result5 !== null) {
                                var result6 = [];
                                var result11 = parse_WS();
                                while (result11 !== null) {
                                    result6.push(result11);
                                    var result11 = parse_WS();
                                }
                                if (result6 !== null) {
                                    var result10 = parse_OrderCondition();
                                    if (result10 !== null) {
                                        var result7 = [];
                                        while (result10 !== null) {
                                            result7.push(result10);
                                            var result10 = parse_OrderCondition();
                                        }
                                    } else {
                                        var result7 = null;
                                    }
                                    if (result7 !== null) {
                                        var result8 = [];
                                        var result9 = parse_WS();
                                        while (result9 !== null) {
                                            result8.push(result9);
                                            var result9 = parse_WS();
                                        }
                                        if (result8 !== null) {
                                            var result1 = [result3, result4, result5, result6, result7,
result8];
                                        } else {
                                            var result1 = null;
                                            pos = savedPos1;
                                        }
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                    var result2 = result1 !== null
                        ? (function(os) {
                        return os;
```

```
        })(result1[4])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[22] OrderClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_OrderCondition() {
        var cacheKey = 'OrderCondition@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos2 = pos;
        var savedPos3 = pos;
        if (input.substr(pos, 3) === "ASC") {
            var result21 = "ASC";
            pos += 3;
        } else {
            var result21 = null;
            if (reportMatchFailures) {
                matchFailed("\"ASC\"");
            }
        }
        if (result21 !== null) {
            var result12 = result21;
        } else {
            if (input.substr(pos, 3) === "asc") {
                var result20 = "asc";
                pos += 3;
            } else {
                var result20 = null;
                if (reportMatchFailures) {
                    matchFailed("\"asc\"");
                }
            }
            if (result20 !== null) {
                var result12 = result20;
            } else {
                if (input.substr(pos, 4) === "DESC") {
                    var result19 = "DESC";
                    pos += 4;
                } else {
                    var result19 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"DESC\"");
                    }
                }
                if (result19 !== null) {
                    var result12 = result19;
```

```
            } else {
                if (input.substr(pos, 4) === "desc") {
                    var result18 = "desc";
                    pos += 4;
                } else {
                    var result18 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"desc\"");
                    }
                }
                if (result18 !== null) {
                    var result12 = result18;
                } else {
                    var result12 = null;;
                };
            };
        };
    }
    if (result12 !== null) {
        var result13 = [];
        var result17 = parse_WS();
        while (result17 !== null) {
            result13.push(result17);
            var result17 = parse_WS();
        }
        if (result13 !== null) {
            var result14 = parse_BracketedExpression();
            if (result14 !== null) {
                var result15 = [];
                var result16 = parse_WS();
                while (result16 !== null) {
                    result15.push(result16);
                    var result16 = parse_WS();
                }
                if (result15 !== null) {
                    var result10 = [result12, result13, result14, result15];
                } else {
                    var result10 = null;
                    pos = savedPos3;
                }
            } else {
                var result10 = null;
                pos = savedPos3;
            }
        } else {
            var result10 = null;
            pos = savedPos3;
        }
    } else {
        var result10 = null;
        pos = savedPos3;
    }
    var result11 = result10 !== null
        ? (function(direction, e) {
        return { direction: direction.toUpperCase(), expression:e };
    })(result10[0], result10[2])
        : null;
    if (result11 !== null) {
        var result9 = result11;
    } else {
        var result9 = null;
        pos = savedPos2;
    }
    if (result9 !== null) {
        var result0 = result9;
    } else {
        var savedPos0 = pos;
```

```
            var savedPos1 = pos;
            var result8 = parse_Constraint();
            if (result8 !== null) {
                var result4 = result8;
            } else {
                var result7 = parse_Var();
                if (result7 !== null) {
                    var result4 = result7;
                } else {
                    var result4 = null;;
                };
            }
            if (result4 !== null) {
                var result5 = [];
                var result6 = parse_WS();
                while (result6 !== null) {
                    result5.push(result6);
                    var result6 = parse_WS();
                }
                if (result5 !== null) {
                    var result2 = [result4, result5];
                } else {
                    var result2 = null;
                    pos = savedPos1;
                }
            } else {
                var result2 = null;
                pos = savedPos1;
            }
            var result3 = result2 !== null
                ? (function(e) {
                if(e.token === 'var') {
                    var e = { token:'expression',
                        expressionType:'atomic',
                        primaryexpression: 'var',
                        value: e };
                }
                return { direction: 'ASC', expression:e };
            })(result2[0])
                : null;
            if (result3 !== null) {
                var result1 = result3;
            } else {
                var result1 = null;
                pos = savedPos0;
            }
            if (result1 !== null) {
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[23] OrderCondition");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_LimitOffsetClauses() {
        var cacheKey = 'LimitOffsetClauses@' + pos;
        var cachedResult = cache[cacheKey];
```

```
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos2 = pos;
var result8 = parse_LimitClause();
if (result8 !== null) {
    var result10 = parse_OffsetClause();
    var result9 = result10 !== null ? result10 : '';
    if (result9 !== null) {
        var result7 = [result8, result9];
    } else {
        var result7 = null;
        pos = savedPos2;
    }
} else {
    var result7 = null;
    pos = savedPos2;
}
if (result7 !== null) {
    var result1 = result7;
} else {
    var savedPos1 = pos;
    var result4 = parse_OffsetClause();
    if (result4 !== null) {
        var result6 = parse_LimitClause();
        var result5 = result6 !== null ? result6 : '';
        if (result5 !== null) {
            var result3 = [result4, result5];
        } else {
            var result3 = null;
            pos = savedPos1;
        }
    } else {
        var result3 = null;
        pos = savedPos1;
    }
    if (result3 !== null) {
        var result1 = result3;
    } else {
        var result1 = null;;
    };
}
var result2 = result1 !== null
    ? (function(cls) {
    var acum = {};
    for(var i=0; i<cls.length; i++) {
        var cl = cls[i];
        if(cl.limit != null) {
            acum['limit'] = cl.limit;
        } else if(cl.offset != null){
            acum['offset'] = cl.offset;
        }
    }

    return acum;
})(result1)
    : null;
if (result2 !== null) {
    var result0 = result2;
} else {
    var result0 = null;
    pos = savedPos0;
```

```
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[24] LimitOffsetClauses");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_LimitClause() {
        var cacheKey = 'LimitClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 5) === "LIMIT") {
            var result10 = "LIMIT";
            pos += 5;
        } else {
            var result10 = null;
            if (reportMatchFailures) {
                matchFailed("\"LIMIT\"");
            }
        }
        if (result10 !== null) {
            var result3 = result10;
        } else {
            if (input.substr(pos, 5) === "limit") {
                var result9 = "limit";
                pos += 5;
            } else {
                var result9 = null;
                if (reportMatchFailures) {
                    matchFailed("\"limit\"");
                }
            }
            if (result9 !== null) {
                var result3 = result9;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result8 = parse_WS();
            while (result8 !== null) {
                result4.push(result8);
                var result8 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = parse_INTEGER();
                if (result5 !== null) {
                    var result6 = [];
                    var result7 = parse_WS();
                    while (result7 !== null) {
                        result6.push(result7);
                        var result7 = parse_WS();
```

```
                    }
                    if (result6 !== null) {
                        var result1 = [result3, result4, result5, result6];
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(i) {
            return { limit:parseInt(i.value) };
        })(result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[25] LimitClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_OffsetClause() {
        var cacheKey = 'OffsetClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 6) === "OFFSET") {
            var result10 = "OFFSET";
            pos += 6;
        } else {
            var result10 = null;
            if (reportMatchFailures) {
                matchFailed("\"OFFSET\"");
            }
        }
        if (result10 !== null) {
            var result3 = result10;
        } else {
            if (input.substr(pos, 6) === "offset") {
```

```
            var result9 = "offset";
            pos += 6;
        } else {
            var result9 = null;
            if (reportMatchFailures) {
                matchFailed("\"offset\"");
            }
        }
        if (result9 !== null) {
            var result3 = result9;
        } else {
            var result3 = null;;
        };
    }
    if (result3 !== null) {
        var result4 = [];
        var result8 = parse_WS();
        while (result8 !== null) {
            result4.push(result8);
            var result8 = parse_WS();
        }
        if (result4 !== null) {
            var result5 = parse_INTEGER();
            if (result5 !== null) {
                var result6 = [];
                var result7 = parse_WS();
                while (result7 !== null) {
                    result6.push(result7);
                    var result7 = parse_WS();
                }
                if (result6 !== null) {
                    var result1 = [result3, result4, result5, result6];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(i) {
        return { offset:parseInt(i.value) };
    })(result1[2])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[26] OffsetClause");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
```

```
        };
        return result0;
    }

    function parse_BindingsClause() {
        var cacheKey = 'BindingsClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        if (input.substr(pos, 8) === "BINDINGS") {
            var result2 = "BINDINGS";
            pos += 8;
        } else {
            var result2 = null;
            if (reportMatchFailures) {
                matchFailed("\"BINDINGS\"");
            }
        }
        if (result2 !== null) {
            var result3 = [];
            var result14 = parse_Var();
            while (result14 !== null) {
                result3.push(result14);
                var result14 = parse_Var();
            }
            if (result3 !== null) {
                if (input.substr(pos, 1) === "{") {
                    var result4 = "{";
                    pos += 1;
                } else {
                    var result4 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"{\"");
                    }
                }
                if (result4 !== null) {
                    var result5 = [];
                    var savedPos1 = pos;
                    if (input.substr(pos, 1) === "(") {
                        var result10 = "(";
                        pos += 1;
                    } else {
                        var result10 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"(\"");
                        }
                    }
                    if (result10 !== null) {
                        var result13 = parse_BindingValue();
                        if (result13 !== null) {
                            var result11 = [];
                            while (result13 !== null) {
                                result11.push(result13);
                                var result13 = parse_BindingValue();
                            }
                        } else {
                            var result11 = null;
                        }
                        if (result11 !== null) {
                            if (input.substr(pos, 1) === ")") {
                                var result12 = ")";
```

```
                    pos += 1;
                } else {
                    var result12 = null;
                    if (reportMatchFailures) {
                        matchFailed("\")\"");
                    }
                }
                if (result12 !== null) {
                    var result9 = [result10, result11, result12];
                } else {
                    var result9 = null;
                    pos = savedPos1;
                }
            } else {
                var result9 = null;
                pos = savedPos1;
            }
        } else {
            var result9 = null;
            pos = savedPos1;
        }
        if (result9 !== null) {
            var result7 = result9;
        } else {
            var result8 = parse_NIL();
            if (result8 !== null) {
                var result7 = result8;
            } else {
                var result7 = null;;
            };
        }
        while (result7 !== null) {
            result5.push(result7);
            var savedPos1 = pos;
            if (input.substr(pos, 1) === "(") {
                var result10 = "(";
                pos += 1;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("\"(\"");
                }
            }
            if (result10 !== null) {
                var result13 = parse_BindingValue();
                if (result13 !== null) {
                    var result11 = [];
                    while (result13 !== null) {
                        result11.push(result13);
                        var result13 = parse_BindingValue();
                    }
                } else {
                    var result11 = null;
                }
                if (result11 !== null) {
                    if (input.substr(pos, 1) === ")") {
                        var result12 = ")";
                        pos += 1;
                    } else {
                        var result12 = null;
                        if (reportMatchFailures) {
                            matchFailed("\")\"");
                        }
                    }
                    if (result12 !== null) {
                        var result9 = [result10, result11, result12];
                    } else {
```

```
                                    var result9 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result9 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result9 = null;
                            pos = savedPos1;
                        }
                        if (result9 !== null) {
                            var result7 = result9;
                        } else {
                            var result8 = parse_NIL();
                            if (result8 !== null) {
                                var result7 = result8;
                            } else {
                                var result7 = null;;
                            };
                        }
                    }
                    if (result5 !== null) {
                        if (input.substr(pos, 1) === "}") {
                            var result6 = "}";
                            pos += 1;
                        } else {
                            var result6 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"}\"");
                            }
                        }
                        if (result6 !== null) {
                            var result1 = [result2, result3, result4, result5, result6];
                        } else {
                            var result1 = null;
                            pos = savedPos0;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos0;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos0;
                }
            } else {
                var result1 = null;
                pos = savedPos0;
            }
        } else {
            var result1 = null;
            pos = savedPos0;
        }
        var result0 = result1 !== null ? result1 : '';
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[27] BindingsClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }
```

```
        function parse_BindingValue() {
            var cacheKey = 'BindingValue@' + pos;
            var cachedResult = cache[cacheKey];
            if (cachedResult) {
                pos = cachedResult.nextPos;
                return cachedResult.result;
            }

            var savedReportMatchFailures = reportMatchFailures;
            reportMatchFailures = false;
            var result5 = parse_IRIref();
            if (result5 !== null) {
                var result0 = result5;
            } else {
                var result4 = parse_RDFLiteral();
                if (result4 !== null) {
                    var result0 = result4;
                } else {
                    var result3 = parse_NumericLiteral();
                    if (result3 !== null) {
                        var result0 = result3;
                    } else {
                        var result2 = parse_BooleanLiteral();
                        if (result2 !== null) {
                            var result0 = result2;
                        } else {
                            if (input.substr(pos, 5) === "UNDEF") {
                                var result1 = "UNDEF";
                                pos += 5;
                            } else {
                                var result1 = null;
                                if (reportMatchFailures) {
                                    matchFailed("\"UNDEF\"");
                                }
                            }
                            if (result1 !== null) {
                                var result0 = result1;
                            } else {
                                var result0 = null;;
                            };
                        };
                    };
                };
            }
            reportMatchFailures = savedReportMatchFailures;
            if (reportMatchFailures && result0 === null) {
                matchFailed("[28] BindingValue");
            }

            cache[cacheKey] = {
                nextPos: pos,
                result:  result0
            };
            return result0;
        }

        function parse_Update() {
            var cacheKey = 'Update@' + pos;
            var cachedResult = cache[cacheKey];
            if (cachedResult) {
                pos = cachedResult.nextPos;
                return cachedResult.result;
            }

            var savedReportMatchFailures = reportMatchFailures;
            reportMatchFailures = false;
            var savedPos0 = pos;
```

```
            var savedPos1 = pos;
            var result3 = parse_Prologue();
            if (result3 !== null) {
                var result4 = [];
                var result15 = parse_WS();
                while (result15 !== null) {
                    result4.push(result15);
                    var result15 = parse_WS();
                }
                if (result4 !== null) {
                    var result5 = parse_Update1();
                    if (result5 !== null) {
                        var savedPos2 = pos;
                        var result8 = [];
                        var result14 = parse_WS();
                        while (result14 !== null) {
                            result8.push(result14);
                            var result14 = parse_WS();
                        }
                        if (result8 !== null) {
                            if (input.substr(pos, 1) === ";") {
                                var result9 = ";";
                                pos += 1;
                            } else {
                                var result9 = null;
                                if (reportMatchFailures) {
                                    matchFailed("\";\"");
                                }
                            }
                            if (result9 !== null) {
                                var result10 = [];
                                var result13 = parse_WS();
                                while (result13 !== null) {
                                    result10.push(result13);
                                    var result13 = parse_WS();
                                }
                                if (result10 !== null) {
                                    var result12 = parse_Update();
                                    var result11 = result12 !== null ? result12 : '';
                                    if (result11 !== null) {
                                        var result7 = [result8, result9, result10, result11];
                                    } else {
                                        var result7 = null;
                                        pos = savedPos2;
                                    }
                                } else {
                                    var result7 = null;
                                    pos = savedPos2;
                                }
                            } else {
                                var result7 = null;
                                pos = savedPos2;
                            }
                        } else {
                            var result7 = null;
                            pos = savedPos2;
                        }
                        var result6 = result7 !== null ? result7 : '';
                        if (result6 !== null) {
                            var result1 = [result3, result4, result5, result6];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
```

```
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(p, u, us) {

            var query = {};
            query.token = 'query';
            query.kind = 'update'
            query.prologue = p;

            var units = [u];

            if(us.length != null && us[3] != null && us[3].units != null) {
                units = units.concat(us[3].units);
            }

            query.units = units;
            return query;
        })(result1[0], result1[2], result1[3])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[30] Update");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Update1() {
        var cacheKey = 'Update1@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result8 = parse_Load();
        if (result8 !== null) {
            var result0 = result8;
        } else {
            var result7 = parse_Clear();
            if (result7 !== null) {
                var result0 = result7;
            } else {
                var result6 = parse_Drop();
                if (result6 !== null) {
                    var result0 = result6;
                } else {
```

```
                    var result5 = parse_Create();
                    if (result5 !== null) {
                        var result0 = result5;
                    } else {
                        var result4 = parse_InsertData();
                        if (result4 !== null) {
                            var result0 = result4;
                        } else {
                            var result3 = parse_DeleteData();
                            if (result3 !== null) {
                                var result0 = result3;
                            } else {
                                var result2 = parse_DeleteWhere();
                                if (result2 !== null) {
                                    var result0 = result2;
                                } else {
                                    var result1 = parse_Modify();
                                    if (result1 !== null) {
                                        var result0 = result1;
                                    } else {
                                        var result0 = null;;
                                    };
                                };
                            };
                        };
                    };
                };
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[31] Update1");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Load() {
        var cacheKey = 'Load@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 4) === "LOAD") {
            var result18 = "LOAD";
            pos += 4;
        } else {
            var result18 = null;
            if (reportMatchFailures) {
                matchFailed("\"LOAD\"");
            }
        }
        if (result18 !== null) {
            var result3 = result18;
        } else {
            if (input.substr(pos, 4) === "load") {
                var result17 = "load";
```

```
                pos += 4;
            } else {
                var result17 = null;
                if (reportMatchFailures) {
                    matchFailed("\"load\"");
                }
            }
            if (result17 !== null) {
                var result3 = result17;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result16 = parse_WS();
            while (result16 !== null) {
                result4.push(result16);
                var result16 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = parse_IRIref();
                if (result5 !== null) {
                    var result6 = [];
                    var result15 = parse_WS();
                    while (result15 !== null) {
                        result6.push(result15);
                        var result15 = parse_WS();
                    }
                    if (result6 !== null) {
                        var savedPos2 = pos;
                        if (input.substr(pos, 4) === "INTO") {
                            var result14 = "INTO";
                            pos += 4;
                        } else {
                            var result14 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"INTO\"");
                            }
                        }
                        if (result14 !== null) {
                            var result9 = result14;
                        } else {
                            if (input.substr(pos, 4) === "into") {
                                var result13 = "into";
                                pos += 4;
                            } else {
                                var result13 = null;
                                if (reportMatchFailures) {
                                    matchFailed("\"into\"");
                                }
                            }
                            if (result13 !== null) {
                                var result9 = result13;
                            } else {
                                var result9 = null;;
                            };
                        }
                        if (result9 !== null) {
                            var result10 = [];
                            var result12 = parse_WS();
                            while (result12 !== null) {
                                result10.push(result12);
                                var result12 = parse_WS();
                            }
                            if (result10 !== null) {
                                var result11 = parse_GraphRef();
```

```
                                if (result11 !== null) {
                                    var result8 = [result9, result10, result11];
                                } else {
                                    var result8 = null;
                                    pos = savedPos2;
                                }
                            } else {
                                var result8 = null;
                                pos = savedPos2;
                            }
                        } else {
                            var result8 = null;
                            pos = savedPos2;
                        }
                        var result7 = result8 !== null ? result8 : '';
                        if (result7 !== null) {
                            var result1 = [result3, result4, result5, result6, result7];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(sg, dg) {
            var query = {};
            query.kind = 'load';
            query.token = 'executableunit'
            query.sourceGraph = sg;
            query.destinyGraph = dg[2];

            return query;
        })(result1[2], result1[4])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[32] Load");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Clear() {
```

```
var cacheKey = 'Clear@' + pos;
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
if (input.substr(pos, 5) === "CLEAR") {
    var result14 = "CLEAR";
    pos += 5;
} else {
    var result14 = null;
    if (reportMatchFailures) {
        matchFailed("\"CLEAR\"");
    }
}
if (result14 !== null) {
    var result3 = result14;
} else {
    if (input.substr(pos, 5) === "clear") {
        var result13 = "clear";
        pos += 5;
    } else {
        var result13 = null;
        if (reportMatchFailures) {
            matchFailed("\"clear\"");
        }
    }
    if (result13 !== null) {
        var result3 = result13;
    } else {
        var result3 = null;;
    };
}
if (result3 !== null) {
    var result4 = [];
    var result12 = parse_WS();
    while (result12 !== null) {
        result4.push(result12);
        var result12 = parse_WS();
    }
    if (result4 !== null) {
        if (input.substr(pos, 6) === "SILENT") {
            var result11 = "SILENT";
            pos += 6;
        } else {
            var result11 = null;
            if (reportMatchFailures) {
                matchFailed("\"SILENT\"");
            }
        }
        if (result11 !== null) {
            var result9 = result11;
        } else {
            if (input.substr(pos, 6) === "silent") {
                var result10 = "silent";
                pos += 6;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("\"silent\"");
                }
            }
```

```
                    if (result10 !== null) {
                        var result9 = result10;
                    } else {
                        var result9 = null;;
                    };
                }
                var result5 = result9 !== null ? result9 : '';
                if (result5 !== null) {
                    var result6 = [];
                    var result8 = parse_WS();
                    while (result8 !== null) {
                        result6.push(result8);
                        var result8 = parse_WS();
                    }
                    if (result6 !== null) {
                        var result7 = parse_GraphRefAll();
                        if (result7 !== null) {
                            var result1 = [result3, result4, result5, result6, result7];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(ref) {
            var query = {};
            query.kind = 'clear';
            query.token = 'executableunit'
            query.destinyGraph = ref;

            return query;
        })(result1[4])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[33] Clear");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Drop() {
```

```
var cacheKey = 'Drop@' + pos;
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
if (input.substr(pos, 4) === "DROP") {
    var result14 = "DROP";
    pos += 4;
} else {
    var result14 = null;
    if (reportMatchFailures) {
        matchFailed("\"DROP\"");
    }
}
if (result14 !== null) {
    var result3 = result14;
} else {
    if (input.substr(pos, 4) === "drop") {
        var result13 = "drop";
        pos += 4;
    } else {
        var result13 = null;
        if (reportMatchFailures) {
            matchFailed("\"drop\"");
        }
    }
    if (result13 !== null) {
        var result3 = result13;
    } else {
        var result3 = null;;
    };
}
if (result3 !== null) {
    var result4 = [];
    var result12 = parse_WS();
    while (result12 !== null) {
        result4.push(result12);
        var result12 = parse_WS();
    }
    if (result4 !== null) {
        if (input.substr(pos, 6) === "SILENT") {
            var result11 = "SILENT";
            pos += 6;
        } else {
            var result11 = null;
            if (reportMatchFailures) {
                matchFailed("\"SILENT\"");
            }
        }
        if (result11 !== null) {
            var result9 = result11;
        } else {
            if (input.substr(pos, 6) === "silent") {
                var result10 = "silent";
                pos += 6;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("\"silent\"");
                }
            }
```

```
                    if (result10 !== null) {
                        var result9 = result10;
                    } else {
                        var result9 = null;;
                    };
                }
                var result5 = result9 !== null ? result9 : '';
                if (result5 !== null) {
                    var result6 = [];
                    var result8 = parse_WS();
                    while (result8 !== null) {
                        result6.push(result8);
                        var result8 = parse_WS();
                    }
                    if (result6 !== null) {
                        var result7 = parse_GraphRefAll();
                        if (result7 !== null) {
                            var result1 = [result3, result4, result5, result6, result7];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(ref) {
            var query = {};
            query.kind = 'drop';
            query.token = 'executableunit'
            query.destinyGraph = ref;

            return query;
        })(result1[4])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[34] Drop");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Create() {
```

```
var cacheKey = 'Create@' + pos;
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
if (input.substr(pos, 6) === "CREATE") {
    var result14 = "CREATE";
    pos += 6;
} else {
    var result14 = null;
    if (reportMatchFailures) {
        matchFailed("\"CREATE\"");
    }
}
if (result14 !== null) {
    var result3 = result14;
} else {
    if (input.substr(pos, 6) === "create") {
        var result13 = "create";
        pos += 6;
    } else {
        var result13 = null;
        if (reportMatchFailures) {
            matchFailed("\"create\"");
        }
    }
    if (result13 !== null) {
        var result3 = result13;
    } else {
        var result3 = null;;
    };
}
if (result3 !== null) {
    var result4 = [];
    var result12 = parse_WS();
    while (result12 !== null) {
        result4.push(result12);
        var result12 = parse_WS();
    }
    if (result4 !== null) {
        if (input.substr(pos, 6) === "SILENT") {
            var result11 = "SILENT";
            pos += 6;
        } else {
            var result11 = null;
            if (reportMatchFailures) {
                matchFailed("\"SILENT\"");
            }
        }
        if (result11 !== null) {
            var result9 = result11;
        } else {
            if (input.substr(pos, 6) === "silent") {
                var result10 = "silent";
                pos += 6;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("\"silent\"");
                }
            }
```

```
                    if (result10 !== null) {
                        var result9 = result10;
                    } else {
                        var result9 = null;;
                    };
                }
                var result5 = result9 !== null ? result9 : '';
                if (result5 !== null) {
                    var result6 = [];
                    var result8 = parse_WS();
                    while (result8 !== null) {
                        result6.push(result8);
                        var result8 = parse_WS();
                    }
                    if (result6 !== null) {
                        var result7 = parse_GraphRef();
                        if (result7 !== null) {
                            var result1 = [result3, result4, result5, result6, result7];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(ref) {
            var query = {};
            query.kind = 'create';
            query.token = 'executableunit'
            query.destinyGraph = ref;

            return query;
        })(result1[4])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[35] Create");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_InsertData() {
```

```
var cacheKey = 'InsertData@' + pos;
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
if (input.substr(pos, 6) === "INSERT") {
    var result13 = "INSERT";
    pos += 6;
} else {
    var result13 = null;
    if (reportMatchFailures) {
        matchFailed("\"INSERT\"");
    }
}
if (result13 !== null) {
    var result3 = result13;
} else {
    if (input.substr(pos, 6) === "insert") {
        var result12 = "insert";
        pos += 6;
    } else {
        var result12 = null;
        if (reportMatchFailures) {
            matchFailed("\"insert\"");
        }
    }
    if (result12 !== null) {
        var result3 = result12;
    } else {
        var result3 = null;;
    };
}
if (result3 !== null) {
    var result4 = [];
    var result11 = parse_WS();
    while (result11 !== null) {
        result4.push(result11);
        var result11 = parse_WS();
    }
    if (result4 !== null) {
        if (input.substr(pos, 4) === "DATA") {
            var result10 = "DATA";
            pos += 4;
        } else {
            var result10 = null;
            if (reportMatchFailures) {
                matchFailed("\"DATA\"");
            }
        }
        if (result10 !== null) {
            var result5 = result10;
        } else {
            if (input.substr(pos, 4) === "data") {
                var result9 = "data";
                pos += 4;
            } else {
                var result9 = null;
                if (reportMatchFailures) {
                    matchFailed("\"data\"");
                }
            }
```

```
                    if (result9 !== null) {
                        var result5 = result9;
                    } else {
                        var result5 = null;;
                    };
                }
                if (result5 !== null) {
                    var result6 = [];
                    var result8 = parse_WS();
                    while (result8 !== null) {
                        result6.push(result8);
                        var result8 = parse_WS();
                    }
                    if (result6 !== null) {
                        var result7 = parse_QuadData();
                        if (result7 !== null) {
                            var result1 = [result3, result4, result5, result6, result7];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(qs) {
            var query = {};
            query.kind = 'insertdata';
            query.token = 'executableunit'
            query.quads = qs;

            return query;
        })(result1[4])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[36] InsertData");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DeleteData() {
        var cacheKey = 'DeleteData@' + pos;
```

```
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
if (input.substr(pos, 6) === "DELETE") {
    var result11 = "DELETE";
    pos += 6;
} else {
    var result11 = null;
    if (reportMatchFailures) {
        matchFailed("\"DELETE\"");
    }
}
if (result11 !== null) {
    var result3 = result11;
} else {
    if (input.substr(pos, 6) === "delete") {
        var result10 = "delete";
        pos += 6;
    } else {
        var result10 = null;
        if (reportMatchFailures) {
            matchFailed("\"delete\"");
        }
    }
    if (result10 !== null) {
        var result3 = result10;
    } else {
        var result3 = null;;
    };
}
if (result3 !== null) {
    var result4 = [];
    var result9 = parse_WS();
    while (result9 !== null) {
        result4.push(result9);
        var result9 = parse_WS();
    }
    if (result4 !== null) {
        if (input.substr(pos, 4) === "DATA") {
            var result8 = "DATA";
            pos += 4;
        } else {
            var result8 = null;
            if (reportMatchFailures) {
                matchFailed("\"DATA\"");
            }
        }
        if (result8 !== null) {
            var result5 = result8;
        } else {
            if (input.substr(pos, 4) === "data") {
                var result7 = "data";
                pos += 4;
            } else {
                var result7 = null;
                if (reportMatchFailures) {
                    matchFailed("\"data\"");
                }
            }
            if (result7 !== null) {
```

```
                    var result5 = result7;
                } else {
                    var result5 = null;;
                };
            }
            if (result5 !== null) {
                var result6 = parse_QuadData();
                if (result6 !== null) {
                    var result1 = [result3, result4, result5, result6];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(qs) {
        var query = {};
        query.kind = 'deletedata';
        query.token = 'executableunit'
        query.quads = qs;

        return query;
    })(result1[3])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[37] DeleteData");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_DeleteWhere() {
    var cacheKey = 'DeleteWhere@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    var savedPos1 = pos;
    if (input.substr(pos, 6) === "DELETE") {
        var result13 = "DELETE";
```

```
            pos += 6;
        } else {
            var result13 = null;
            if (reportMatchFailures) {
                matchFailed("\"DELETE\"");
            }
        }
        if (result13 !== null) {
            var result3 = result13;
        } else {
            if (input.substr(pos, 6) === "delete") {
                var result12 = "delete";
                pos += 6;
            } else {
                var result12 = null;
                if (reportMatchFailures) {
                    matchFailed("\"delete\"");
                }
            }
            if (result12 !== null) {
                var result3 = result12;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result11 = parse_WS();
            while (result11 !== null) {
                result4.push(result11);
                var result11 = parse_WS();
            }
            if (result4 !== null) {
                if (input.substr(pos, 5) === "WHERE") {
                    var result10 = "WHERE";
                    pos += 5;
                } else {
                    var result10 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"WHERE\"");
                    }
                }
                if (result10 !== null) {
                    var result5 = result10;
                } else {
                    if (input.substr(pos, 5) === "where") {
                        var result9 = "where";
                        pos += 5;
                    } else {
                        var result9 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"where\"");
                        }
                    }
                    if (result9 !== null) {
                        var result5 = result9;
                    } else {
                        var result5 = null;;
                    };
                }
                if (result5 !== null) {
                    var result6 = [];
                    var result8 = parse_WS();
                    while (result8 !== null) {
                        result6.push(result8);
                        var result8 = parse_WS();
                    }
```

```
                            if (result6 !== null) {
                                var result7 = parse_GroupGraphPattern();
                                if (result7 !== null) {
                                    var result1 = [result3, result4, result5, result6, result7];
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function(p) {
                    var query = {};
                    query.kind = 'modify';
                    query.pattern = p;
                    query.with = null;
                    query.using = null;

                    var quads = [];


                    var patternsCollection = p.patterns[0];
                    if(patternsCollection.triplesContext == null && patternsCollection.patterns!
=null) {
                        patternsCollection = patternsCollection.patterns[0].triplesContext;
                    } else {
                        patternsCollection = patternsCollection.triplesContext;
                    }

                    for(var i=0; i<patternsCollection.length; i++) {
                        var quad = {};
                        var contextQuad = patternsCollection[i];

                        quad['subject'] = contextQuad['subject'];
                        quad['predicate'] = contextQuad['predicate'];
                        quad['object'] = contextQuad['object'];
                        quad['graph'] = contextQuad['graph'];

                        quads.push(quad);
                    }

                    query.delete = quads;

                    return query;
                })(result1[4])
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
```

```
        if (reportMatchFailures && result0 === null) {
            matchFailed("[38] DeleteWhere");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Modify() {
        var cacheKey = 'Modify@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var savedPos3 = pos;
        if (input.substr(pos, 4) === "WITH") {
            var result34 = "WITH";
            pos += 4;
        } else {
            var result34 = null;
            if (reportMatchFailures) {
                matchFailed("\"WITH\"");
            }
        }
        if (result34 !== null) {
            var result29 = result34;
        } else {
            if (input.substr(pos, 4) === "with") {
                var result33 = "with";
                pos += 4;
            } else {
                var result33 = null;
                if (reportMatchFailures) {
                    matchFailed("\"with\"");
                }
            }
            if (result33 !== null) {
                var result29 = result33;
            } else {
                var result29 = null;;
            };
        }
        if (result29 !== null) {
            var result30 = [];
            var result32 = parse_WS();
            while (result32 !== null) {
                result30.push(result32);
                var result32 = parse_WS();
            }
            if (result30 !== null) {
                var result31 = parse_IRIref();
                if (result31 !== null) {
                    var result28 = [result29, result30, result31];
                } else {
                    var result28 = null;
                    pos = savedPos3;
                }
            } else {
```

```
            var result28 = null;
            pos = savedPos3;
        }
    } else {
        var result28 = null;
        pos = savedPos3;
    }
    var result3 = result28 !== null ? result28 : '';
    if (result3 !== null) {
        var result4 = [];
        var result27 = parse_WS();
        while (result27 !== null) {
            result4.push(result27);
            var result27 = parse_WS();
        }
        if (result4 !== null) {
            var savedPos2 = pos;
            var result22 = parse_DeleteClause();
            if (result22 !== null) {
                var result23 = [];
                var result26 = parse_WS();
                while (result26 !== null) {
                    result23.push(result26);
                    var result26 = parse_WS();
                }
                if (result23 !== null) {
                    var result25 = parse_InsertClause();
                    var result24 = result25 !== null ? result25 : '';
                    if (result24 !== null) {
                        var result21 = [result22, result23, result24];
                    } else {
                        var result21 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result21 = null;
                    pos = savedPos2;
                }
            } else {
                var result21 = null;
                pos = savedPos2;
            }
            if (result21 !== null) {
                var result5 = result21;
            } else {
                var result20 = parse_InsertClause();
                if (result20 !== null) {
                    var result5 = result20;
                } else {
                    var result5 = null;;
                };
            }
            if (result5 !== null) {
                var result6 = [];
                var result19 = parse_WS();
                while (result19 !== null) {
                    result6.push(result19);
                    var result19 = parse_WS();
                }
                if (result6 !== null) {
                    var result7 = [];
                    var result18 = parse_UsingClause();
                    while (result18 !== null) {
                        result7.push(result18);
                        var result18 = parse_UsingClause();
                    }
                    if (result7 !== null) {
```

```
                                    var result8 = [];
                                    var result17 = parse_WS();
                                    while (result17 !== null) {
                                        result8.push(result17);
                                        var result17 = parse_WS();
                                    }
                                    if (result8 !== null) {
                                        if (input.substr(pos, 5) === "WHERE") {
                                            var result16 = "WHERE";
                                            pos += 5;
                                        } else {
                                            var result16 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\"WHERE\"");
                                            }
                                        }
                                        if (result16 !== null) {
                                            var result9 = result16;
                                        } else {
                                            if (input.substr(pos, 5) === "where") {
                                                var result15 = "where";
                                                pos += 5;
                                            } else {
                                                var result15 = null;
                                                if (reportMatchFailures) {
                                                    matchFailed("\"where\"");
                                                }
                                            }
                                            if (result15 !== null) {
                                                var result9 = result15;
                                            } else {
                                                var result9 = null;;
                                            };
                                        }
                                        if (result9 !== null) {
                                            var result10 = [];
                                            var result14 = parse_WS();
                                            while (result14 !== null) {
                                                result10.push(result14);
                                                var result14 = parse_WS();
                                            }
                                            if (result10 !== null) {
                                                var result11 = parse_GroupGraphPattern();
                                                if (result11 !== null) {
                                                    var result12 = [];
                                                    var result13 = parse_WS();
                                                    while (result13 !== null) {
                                                        result12.push(result13);
                                                        var result13 = parse_WS();
                                                    }
                                                    if (result12 !== null) {
                                                        var result1 = [result3, result4, result5,
result6, result7, result8, result9, result10, result11, result12];
                                                    } else {
                                                        var result1 = null;
                                                        pos = savedPos1;
                                                    }
                                                } else {
                                                    var result1 = null;
                                                    pos = savedPos1;
                                                }
                                            } else {
                                                var result1 = null;
                                                pos = savedPos1;
                                            }
                                        } else {
                                            var result1 = null;
```

```
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function(wg, dic, uc, p) {
                    var query = {};
                    query.kind = 'modify';

                    if(wg != "") {
                        query.with = wg[2];
                    } else {
                        query.with = null;
                    }


                    if(dic.length === 3 && dic[2] === '') {
                        query.delete = dic[0];
                        query.insert = null;
                    } else if(dic.length === 3 && dic[0].length != null && dic[1].length != null &&
dic[2].length != null) {
                        query.delete = dic[0];
                        query.insert = dic[2];
                    } else  {
                        query.insert = dic;
                        query.delete = null;
                    }

                    if(uc != '') {
                        query.using = uc;
                    }

                    query.pattern = p;

                    return query;
                })(result1[0], result1[2], result1[4], result1[8])
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
```

```
            matchFailed("[39] Modify");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DeleteClause() {
        var cacheKey = 'DeleteClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 6) === "DELETE") {
            var result6 = "DELETE";
            pos += 6;
        } else {
            var result6 = null;
            if (reportMatchFailures) {
                matchFailed("\"DELETE\"");
            }
        }
        if (result6 !== null) {
            var result3 = result6;
        } else {
            if (input.substr(pos, 6) === "delete") {
                var result5 = "delete";
                pos += 6;
            } else {
                var result5 = null;
                if (reportMatchFailures) {
                    matchFailed("\"delete\"");
                }
            }
            if (result5 !== null) {
                var result3 = result5;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = parse_QuadPattern();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(q) {
            return q;
        })(result1[1])
            : null;
        if (result2 !== null) {
```

```
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[40] DeleteClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_InsertClause() {
        var cacheKey = 'InsertClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 6) === "INSERT") {
            var result6 = "INSERT";
            pos += 6;
        } else {
            var result6 = null;
            if (reportMatchFailures) {
                matchFailed("\"INSERT\"");
            }
        }
        if (result6 !== null) {
            var result3 = result6;
        } else {
            if (input.substr(pos, 6) === "insert") {
                var result5 = "insert";
                pos += 6;
            } else {
                var result5 = null;
                if (reportMatchFailures) {
                    matchFailed("\"insert\"");
                }
            }
            if (result5 !== null) {
                var result3 = result5;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = parse_QuadPattern();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
```

```
        }
        var result2 = result1 !== null
            ? (function(q) {
            return q;
        })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[41] InsertClause");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_UsingClause() {
        var cacheKey = 'UsingClause@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result18 = parse_WS();
        while (result18 !== null) {
            result3.push(result18);
            var result18 = parse_WS();
        }
        if (result3 !== null) {
            if (input.substr(pos, 5) === "USING") {
                var result17 = "USING";
                pos += 5;
            } else {
                var result17 = null;
                if (reportMatchFailures) {
                    matchFailed("\"USING\"");
                }
            }
            if (result17 !== null) {
                var result4 = result17;
            } else {
                if (input.substr(pos, 5) === "using") {
                    var result16 = "using";
                    pos += 5;
                } else {
                    var result16 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"using\"");
                    }
                }
                if (result16 !== null) {
                    var result4 = result16;
                } else {
```

```
                var result4 = null;;
            };
        }
        if (result4 !== null) {
            var result5 = [];
            var result15 = parse_WS();
            while (result15 !== null) {
                result5.push(result15);
                var result15 = parse_WS();
            }
            if (result5 !== null) {
                var result14 = parse_IRIref();
                if (result14 !== null) {
                    var result6 = result14;
                } else {
                    var savedPos2 = pos;
                    if (input.substr(pos, 5) === "NAMED") {
                        var result13 = "NAMED";
                        pos += 5;
                    } else {
                        var result13 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"NAMED\"");
                        }
                    }
                    if (result13 !== null) {
                        var result8 = result13;
                    } else {
                        if (input.substr(pos, 5) === "named") {
                            var result12 = "named";
                            pos += 5;
                        } else {
                            var result12 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"named\"");
                            }
                        }
                        if (result12 !== null) {
                            var result8 = result12;
                        } else {
                            var result8 = null;;
                        };
                    }
                    if (result8 !== null) {
                        var result9 = [];
                        var result11 = parse_WS();
                        while (result11 !== null) {
                            result9.push(result11);
                            var result11 = parse_WS();
                        }
                        if (result9 !== null) {
                            var result10 = parse_IRIref();
                            if (result10 !== null) {
                                var result7 = [result8, result9, result10];
                            } else {
                                var result7 = null;
                                pos = savedPos2;
                            }
                        } else {
                            var result7 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result7 = null;
                        pos = savedPos2;
                    }
                    if (result7 !== null) {
```

```
                        var result6 = result7;
                    } else {
                        var result6 = null;;
                    };
                }
                if (result6 !== null) {
                    var result1 = [result3, result4, result5, result6];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(g) {
        if(g.length!=null) {
            return {kind: 'named', uri: g[2]};
        } else {
            return {kind: 'default', uri: g};
        }
    })(result1[3])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[42] UsingClause");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_GraphRef() {
    var cacheKey = 'GraphRef@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    var savedPos1 = pos;
    if (input.substr(pos, 5) === "GRAPH") {
        var result8 = "GRAPH";
        pos += 5;
    } else {
        var result8 = null;
```

```
            if (reportMatchFailures) {
                matchFailed("\"GRAPH\"");
            }
        }
        if (result8 !== null) {
            var result3 = result8;
        } else {
            if (input.substr(pos, 5) === "graph") {
                var result7 = "graph";
                pos += 5;
            } else {
                var result7 = null;
                if (reportMatchFailures) {
                    matchFailed("\"graph\"");
                }
            }
            if (result7 !== null) {
                var result3 = result7;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result6 = parse_WS();
            while (result6 !== null) {
                result4.push(result6);
                var result6 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = parse_IRIref();
                if (result5 !== null) {
                    var result1 = [result3, result4, result5];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(i) {
            return i;
        })(result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[43] GraphRef");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }
```

```
function parse_GraphRefAll() {
    var cacheKey = 'GraphRefAll@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos3 = pos;
    var result17 = parse_GraphRef();
    var result18 = result17 !== null
        ? (function(g) {
        return g;
    })(result17)
        : null;
    if (result18 !== null) {
        var result16 = result18;
    } else {
        var result16 = null;
        pos = savedPos3;
    }
    if (result16 !== null) {
        var result0 = result16;
    } else {
        var savedPos2 = pos;
        if (input.substr(pos, 7) === "DEFAULT") {
            var result15 = "DEFAULT";
            pos += 7;
        } else {
            var result15 = null;
            if (reportMatchFailures) {
                matchFailed("\"DEFAULT\"");
            }
        }
        if (result15 !== null) {
            var result12 = result15;
        } else {
            if (input.substr(pos, 7) === "default") {
                var result14 = "default";
                pos += 7;
            } else {
                var result14 = null;
                if (reportMatchFailures) {
                    matchFailed("\"default\"");
                }
            }
            if (result14 !== null) {
                var result12 = result14;
            } else {
                var result12 = null;;
            };
        }
        var result13 = result12 !== null
            ? (function() {
            return 'default';
        })()
            : null;
        if (result13 !== null) {
            var result11 = result13;
        } else {
            var result11 = null;
            pos = savedPos2;
        }
        if (result11 !== null) {
```

```
            var result0 = result11;
        } else {
            var savedPos1 = pos;
            if (input.substr(pos, 5) === "NAMED") {
                var result10 = "NAMED";
                pos += 5;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("\"NAMED\"");
                }
            }
            if (result10 !== null) {
                var result7 = result10;
            } else {
                if (input.substr(pos, 5) === "named") {
                    var result9 = "named";
                    pos += 5;
                } else {
                    var result9 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"named\"");
                    }
                }
                if (result9 !== null) {
                    var result7 = result9;
                } else {
                    var result7 = null;;
                };
            }
            var result8 = result7 !== null
                ? (function() {
                return 'named';
            })()
                : null;
            if (result8 !== null) {
                var result6 = result8;
            } else {
                var result6 = null;
                pos = savedPos1;
            }
            if (result6 !== null) {
                var result0 = result6;
            } else {
                var savedPos0 = pos;
                if (input.substr(pos, 3) === "ALL") {
                    var result5 = "ALL";
                    pos += 3;
                } else {
                    var result5 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"ALL\"");
                    }
                }
                if (result5 !== null) {
                    var result2 = result5;
                } else {
                    if (input.substr(pos, 3) === "all") {
                        var result4 = "all";
                        pos += 3;
                    } else {
                        var result4 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"all\"");
                        }
                    }
                    if (result4 !== null) {
```

```
                        var result2 = result4;
                    } else {
                        var result2 = null;;
                    };
                }
                var result3 = result2 !== null
                    ? (function() {
                    return 'all';
                })()
                    : null;
                if (result3 !== null) {
                    var result1 = result3;
                } else {
                    var result1 = null;
                    pos = savedPos0;
                }
                if (result1 !== null) {
                    var result0 = result1;
                } else {
                    var result0 = null;;
                };
            };
        };
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[44] GraphRefAll");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_QuadPattern() {
    var cacheKey = 'QuadPattern@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    var savedPos1 = pos;
    var result3 = [];
    var result13 = parse_WS();
    while (result13 !== null) {
        result3.push(result13);
        var result13 = parse_WS();
    }
    if (result3 !== null) {
        if (input.substr(pos, 1) === "{") {
            var result4 = "{";
            pos += 1;
        } else {
            var result4 = null;
            if (reportMatchFailures) {
                matchFailed("\"{\"");
            }
        }
        if (result4 !== null) {
            var result5 = [];
            var result12 = parse_WS();
```

```
                        while (result12 !== null) {
                            result5.push(result12);
                            var result12 = parse_WS();
                        }
                        if (result5 !== null) {
                            var result6 = parse_Quads();
                            if (result6 !== null) {
                                var result7 = [];
                                var result11 = parse_WS();
                                while (result11 !== null) {
                                    result7.push(result11);
                                    var result11 = parse_WS();
                                }
                                if (result7 !== null) {
                                    if (input.substr(pos, 1) === "}") {
                                        var result8 = "}";
                                        pos += 1;
                                    } else {
                                        var result8 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("\"}\"");
                                        }
                                    }
                                    if (result8 !== null) {
                                        var result9 = [];
                                        var result10 = parse_WS();
                                        while (result10 !== null) {
                                            result9.push(result10);
                                            var result10 = parse_WS();
                                        }
                                        if (result9 !== null) {
                                            var result1 = [result3, result4, result5, result6,
result7, result8, result9];
                                        } else {
                                            var result1 = null;
                                            pos = savedPos1;
                                        }
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                    var result2 = result1 !== null
                        ? (function(qs) {
                        return qs.quadsContext;
                    })(result1[3])
                        : null;
                    if (result2 !== null) {
```

```
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[45] QuadPattern");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_QuadData() {
        var cacheKey = 'QuadData@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result13 = parse_WS();
        while (result13 !== null) {
            result3.push(result13);
            var result13 = parse_WS();
        }
        if (result3 !== null) {
            if (input.substr(pos, 1) === "{") {
                var result4 = "{";
                pos += 1;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("\"{\"");
                }
            }
            if (result4 !== null) {
                var result5 = [];
                var result12 = parse_WS();
                while (result12 !== null) {
                    result5.push(result12);
                    var result12 = parse_WS();
                }
                if (result5 !== null) {
                    var result6 = parse_Quads();
                    if (result6 !== null) {
                        var result7 = [];
                        var result11 = parse_WS();
                        while (result11 !== null) {
                            result7.push(result11);
                            var result11 = parse_WS();
                        }
                        if (result7 !== null) {
                            if (input.substr(pos, 1) === "}") {
                                var result8 = "}";
                                pos += 1;
                            } else {
                                var result8 = null;
```

```
                                        if (reportMatchFailures) {
                                            matchFailed("\"}\"");
                                        }
                                    }
                                    if (result8 !== null) {
                                        var result9 = [];
                                        var result10 = parse_WS();
                                        while (result10 !== null) {
                                            result9.push(result10);
                                            var result10 = parse_WS();
                                        }
                                        if (result9 !== null) {
                                            var result1 = [result3, result4, result5, result6,
result7, result8, result9];
                                        } else {
                                            var result1 = null;
                                            pos = savedPos1;
                                        }
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                    var result2 = result1 !== null
                        ? (function(qs) {
                        return qs.quadsContext;
                    })(result1[3])
                        : null;
                    if (result2 !== null) {
                        var result0 = result2;
                    } else {
                        var result0 = null;
                        pos = savedPos0;
                    }
                    reportMatchFailures = savedReportMatchFailures;
                    if (reportMatchFailures && result0 === null) {
                        matchFailed("[46] QuadData");
                    }

                    cache[cacheKey] = {
                        nextPos: pos,
                        result:  result0
                    };
                    return result0;
                }

                function parse_Quads() {
                    var cacheKey = 'Quads@' + pos;
```

```
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
var result11 = parse_TriplesTemplate();
var result3 = result11 !== null ? result11 : '';
if (result3 !== null) {
    var result4 = [];
    var savedPos2 = pos;
    var result6 = parse_QuadsNotTriples();
    if (result6 !== null) {
        if (input.substr(pos, 1) === ".") {
            var result10 = ".";
            pos += 1;
        } else {
            var result10 = null;
            if (reportMatchFailures) {
                matchFailed("\".\"");
            }
        }
        var result7 = result10 !== null ? result10 : '';
        if (result7 !== null) {
            var result9 = parse_TriplesTemplate();
            var result8 = result9 !== null ? result9 : '';
            if (result8 !== null) {
                var result5 = [result6, result7, result8];
            } else {
                var result5 = null;
                pos = savedPos2;
            }
        } else {
            var result5 = null;
            pos = savedPos2;
        }
    } else {
        var result5 = null;
        pos = savedPos2;
    }
    while (result5 !== null) {
        result4.push(result5);
        var savedPos2 = pos;
        var result6 = parse_QuadsNotTriples();
        if (result6 !== null) {
            if (input.substr(pos, 1) === ".") {
                var result10 = ".";
                pos += 1;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("\".\"");
                }
            }
            var result7 = result10 !== null ? result10 : '';
            if (result7 !== null) {
                var result9 = parse_TriplesTemplate();
                var result8 = result9 !== null ? result9 : '';
                if (result8 !== null) {
                    var result5 = [result6, result7, result8];
                } else {
                    var result5 = null;
                    pos = savedPos2;
```

```
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
        }
        if (result4 !== null) {
            var result1 = [result3, result4];
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(ts, qs) {
        var quads = []
        if(ts.triplesContext != null && ts.triplesContext != null) {
            for(var i=0; i<ts.triplesContext.length; i++) {
                var triple = ts.triplesContext[i]
                triple.graph = null;
                quads.push(triple)
            }
        }

        if(qs && qs.length>0 && qs[0].length > 0) {
            quads = quads.concat(qs[0][0].quadsContext);

            if( qs[0][2] != null && qs[0][2].triplesContext != null) {
                for(var i=0; i<qs[0][2].triplesContext.length; i++) {
                    var triple = qs[0][2].triplesContext[i]
                    triple.graph = null;
                    quads.push(triple)
                }
            }
        }

        return {token:'quads',
            quadsContext: quads}
    })(result1[0], result1[1])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[47] Quads");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_QuadsNotTriples() {
    var cacheKey = 'QuadsNotTriples@' + pos;
```

```
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
var result3 = [];
var result22 = parse_WS();
while (result22 !== null) {
    result3.push(result22);
    var result22 = parse_WS();
}
if (result3 !== null) {
    if (input.substr(pos, 5) === "GRAPH") {
        var result21 = "GRAPH";
        pos += 5;
    } else {
        var result21 = null;
        if (reportMatchFailures) {
            matchFailed("\"GRAPH\"");
        }
    }
    if (result21 !== null) {
        var result4 = result21;
    } else {
        if (input.substr(pos, 5) === "graph") {
            var result20 = "graph";
            pos += 5;
        } else {
            var result20 = null;
            if (reportMatchFailures) {
                matchFailed("\"graph\"");
            }
        }
        if (result20 !== null) {
            var result4 = result20;
        } else {
            var result4 = null;;
        };
    }
    if (result4 !== null) {
        var result5 = [];
        var result19 = parse_WS();
        while (result19 !== null) {
            result5.push(result19);
            var result19 = parse_WS();
        }
        if (result5 !== null) {
            var result6 = parse_VarOrIRIref();
            if (result6 !== null) {
                var result7 = [];
                var result18 = parse_WS();
                while (result18 !== null) {
                    result7.push(result18);
                    var result18 = parse_WS();
                }
                if (result7 !== null) {
                    if (input.substr(pos, 1) === "{") {
                        var result8 = "{";
                        pos += 1;
                    } else {
                        var result8 = null;
                        if (reportMatchFailures) {
```

```
                                        matchFailed("\"{\"");
                                    }
                                }
                                if (result8 !== null) {
                                    var result9 = [];
                                    var result17 = parse_WS();
                                    while (result17 !== null) {
                                        result9.push(result17);
                                        var result17 = parse_WS();
                                    }
                                    if (result9 !== null) {
                                        var result16 = parse_TriplesTemplate();
                                        var result10 = result16 !== null ? result16 : '';
                                        if (result10 !== null) {
                                            var result11 = [];
                                            var result15 = parse_WS();
                                            while (result15 !== null) {
                                                result11.push(result15);
                                                var result15 = parse_WS();
                                            }
                                            if (result11 !== null) {
                                                if (input.substr(pos, 1) === "}") {
                                                    var result12 = "}";
                                                    pos += 1;
                                                } else {
                                                    var result12 = null;
                                                    if (reportMatchFailures) {
                                                        matchFailed("\"}\"");
                                                    }
                                                }
                                                if (result12 !== null) {
                                                    var result13 = [];
                                                    var result14 = parse_WS();
                                                    while (result14 !== null) {
                                                        result13.push(result14);
                                                        var result14 = parse_WS();
                                                    }
                                                    if (result13 !== null) {
                                                        var result1 = [result3, result4, result5, result6, result7, result8, result9, result10, result11, result12, result13];
                                                    } else {
                                                        var result1 = null;
                                                        pos = savedPos1;
                                                    }
                                                } else {
                                                    var result1 = null;
                                                    pos = savedPos1;
                                                }
                                            } else {
                                                var result1 = null;
                                                pos = savedPos1;
                                            }
                                        } else {
                                            var result1 = null;
                                            pos = savedPos1;
                                        }
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
```

```
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(g, ts) {
        var quads = []
        for(var i=0; i<ts.triplesContext.length; i++) {
            var triple = ts.triplesContext[i]
            triple.graph = g;
            quads.push(triple)
        }

        return {token:'quadsnottriples',
            quadsContext: quads}
    })(result1[3], result1[7])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[48] QuadsNotTriples");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_TriplesTemplate() {
    var cacheKey = 'TriplesTemplate@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    var savedPos1 = pos;
    var result3 = parse_TriplesSameSubject();
    if (result3 !== null) {
        var savedPos2 = pos;
        var result6 = [];
        var result12 = parse_WS();
        while (result12 !== null) {
            result6.push(result12);
```

```
            var result12 = parse_WS();
        }
        if (result6 !== null) {
            if (input.substr(pos, 1) === ".") {
                var result7 = ".";
                pos += 1;
            } else {
                var result7 = null;
                if (reportMatchFailures) {
                    matchFailed("\".\"");
                }
            }
            if (result7 !== null) {
                var result8 = [];
                var result11 = parse_WS();
                while (result11 !== null) {
                    result8.push(result11);
                    var result11 = parse_WS();
                }
                if (result8 !== null) {
                    var result10 = parse_TriplesTemplate();
                    var result9 = result10 !== null ? result10 : '';
                    if (result9 !== null) {
                        var result5 = [result6, result7, result8, result9];
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
        } else {
            var result5 = null;
            pos = savedPos2;
        }
        var result4 = result5 !== null ? result5 : '';
        if (result4 !== null) {
            var result1 = [result3, result4];
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(b, bs) {
        var triples = b.triplesContext;
        var toTest = null;
        if(typeof(bs) === 'object') {
            if(bs.length != null) {
                if(bs[3].triplesContext!=null) {
                    triples = triples.concat(bs[3].triplesContext);
                }
            }
        }

        return {token:'triplestemplate',
            triplesContext: triples}
    })(result1[0], result1[1])
        : null;
```

```
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[49] TriplesTemplate");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_GroupGraphPattern() {
        var cacheKey = 'GroupGraphPattern@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos2 = pos;
        var savedPos3 = pos;
        if (input.substr(pos, 1) === "{") {
            var result14 = "{";
            pos += 1;
        } else {
            var result14 = null;
            if (reportMatchFailures) {
                matchFailed("\"{\"");
            }
        }
        if (result14 !== null) {
            var result15 = [];
            var result20 = parse_WS();
            while (result20 !== null) {
                result15.push(result20);
                var result20 = parse_WS();
            }
            if (result15 !== null) {
                var result16 = parse_SubSelect();
                if (result16 !== null) {
                    var result17 = [];
                    var result19 = parse_WS();
                    while (result19 !== null) {
                        result17.push(result19);
                        var result19 = parse_WS();
                    }
                    if (result17 !== null) {
                        if (input.substr(pos, 1) === "}") {
                            var result18 = "}";
                            pos += 1;
                        } else {
                            var result18 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"}\"");
                            }
                        }
                        if (result18 !== null) {
                            var result12 = [result14, result15, result16, result17, result18];
```

```
                    } else {
                        var result12 = null;
                        pos = savedPos3;
                    }
                } else {
                    var result12 = null;
                    pos = savedPos3;
                }
            } else {
                var result12 = null;
                pos = savedPos3;
            }
        } else {
            var result12 = null;
            pos = savedPos3;
        }
    } else {
        var result12 = null;
        pos = savedPos3;
    }
    var result13 = result12 !== null
        ? (function(p) {
        return p;
    })(result12[2])
        : null;
    if (result13 !== null) {
        var result11 = result13;
    } else {
        var result11 = null;
        pos = savedPos2;
    }
    if (result11 !== null) {
        var result0 = result11;
    } else {
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "{") {
            var result4 = "{";
            pos += 1;
        } else {
            var result4 = null;
            if (reportMatchFailures) {
                matchFailed("\"{\"");
            }
        }
        if (result4 !== null) {
            var result5 = [];
            var result10 = parse_WS();
            while (result10 !== null) {
                result5.push(result10);
                var result10 = parse_WS();
            }
            if (result5 !== null) {
                var result6 = parse_GroupGraphPatternSub();
                if (result6 !== null) {
                    var result7 = [];
                    var result9 = parse_WS();
                    while (result9 !== null) {
                        result7.push(result9);
                        var result9 = parse_WS();
                    }
                    if (result7 !== null) {
                        if (input.substr(pos, 1) === "}") {
                            var result8 = "}";
                            pos += 1;
                        } else {
                            var result8 = null;
```

```
                                if (reportMatchFailures) {
                                    matchFailed("\"}\"");
                                }
                            }
                            if (result8 !== null) {
                                var result2 = [result4, result5, result6, result7, result8];
                            } else {
                                var result2 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result2 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result2 = null;
                    pos = savedPos1;
                }
                var result3 = result2 !== null
                    ? (function(p) {
                    return p;
                })(result2[2])
                    : null;
                if (result3 !== null) {
                    var result1 = result3;
                } else {
                    var result1 = null;
                    pos = savedPos0;
                }
                if (result1 !== null) {
                    var result0 = result1;
                } else {
                    var result0 = null;;
                };
            }
            reportMatchFailures = savedReportMatchFailures;
            if (reportMatchFailures && result0 === null) {
                matchFailed("[50] GroupGraphPattern");
            }

            cache[cacheKey] = {
                nextPos: pos,
                result:  result0
            };
            return result0;
        }

        function parse_GroupGraphPatternSub() {
            var cacheKey = 'GroupGraphPatternSub@' + pos;
            var cachedResult = cache[cacheKey];
            if (cachedResult) {
                pos = cachedResult.nextPos;
                return cachedResult.result;
            }

            var savedReportMatchFailures = reportMatchFailures;
            reportMatchFailures = false;
            var savedPos0 = pos;
            var savedPos1 = pos;
```

```
                    var result17 = parse_TriplesBlock();
                    var result3 = result17 !== null ? result17 : '';
                    if (result3 !== null) {
                        var result4 = [];
                        var result16 = parse_WS();
                        while (result16 !== null) {
                            result4.push(result16);
                            var result16 = parse_WS();
                        }
                        if (result4 !== null) {
                            var result5 = [];
                            var savedPos2 = pos;
                            var result7 = parse_GraphPatternNotTriples();
                            if (result7 !== null) {
                                var result8 = [];
                                var result15 = parse_WS();
                                while (result15 !== null) {
                                    result8.push(result15);
                                    var result15 = parse_WS();
                                }
                                if (result8 !== null) {
                                    if (input.substr(pos, 1) === ".") {
                                        var result14 = ".";
                                        pos += 1;
                                    } else {
                                        var result14 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("\".\"");
                                        }
                                    }
                                    var result9 = result14 !== null ? result14 : '';
                                    if (result9 !== null) {
                                        var result10 = [];
                                        var result13 = parse_WS();
                                        while (result13 !== null) {
                                            result10.push(result13);
                                            var result13 = parse_WS();
                                        }
                                        if (result10 !== null) {
                                            var result12 = parse_TriplesBlock();
                                            var result11 = result12 !== null ? result12 : '';
                                            if (result11 !== null) {
                                                var result6 = [result7, result8, result9, result10,
result11];
                                            } else {
                                                var result6 = null;
                                                pos = savedPos2;
                                            }
                                        } else {
                                            var result6 = null;
                                            pos = savedPos2;
                                        }
                                    } else {
                                        var result6 = null;
                                        pos = savedPos2;
                                    }
                                } else {
                                    var result6 = null;
                                    pos = savedPos2;
                                }
                                while (result6 !== null) {
                                    result5.push(result6);
                                    var savedPos2 = pos;
```

```
                            var result7 = parse_GraphPatternNotTriples();
                            if (result7 !== null) {
                                var result8 = [];
                                var result15 = parse_WS();
                                while (result15 !== null) {
                                    result8.push(result15);
                                    var result15 = parse_WS();
                                }
                                if (result8 !== null) {
                                    if (input.substr(pos, 1) === ".") {
                                        var result14 = ".";
                                        pos += 1;
                                    } else {
                                        var result14 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("\".\"");
                                        }
                                    }
                                    var result9 = result14 !== null ? result14 : '';
                                    if (result9 !== null) {
                                        var result10 = [];
                                        var result13 = parse_WS();
                                        while (result13 !== null) {
                                            result10.push(result13);
                                            var result13 = parse_WS();
                                        }
                                        if (result10 !== null) {
                                            var result12 = parse_TriplesBlock();
                                            var result11 = result12 !== null ? result12 : '';
                                            if (result11 !== null) {
                                                var result6 = [result7, result8, result9, result10,
result11];

                                            } else {
                                                var result6 = null;
                                                pos = savedPos2;
                                            }
                                        } else {
                                            var result6 = null;
                                            pos = savedPos2;
                                        }
                                    } else {
                                        var result6 = null;
                                        pos = savedPos2;
                                    }
                                } else {
                                    var result6 = null;
                                    pos = savedPos2;
                                }
                            } else {
                                var result6 = null;
                                pos = savedPos2;
                            }
                        }
                        if (result5 !== null) {
                            var result1 = [result3, result4, result5];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
```

```
                    var result2 = result1 !== null
                        ? (function(tb, tbs) {
                        var subpatterns = [];
                        if(tb != null && tb != []) {
                            subpatterns.push(tb);
                        }

                        for(var i=0; i<tbs.length; i++) {
                            for(var j=0; j< tbs[i].length; j++) {
                                if(tbs[i][j].token != null) {
                                    subpatterns.push(tbs[i][j]);
                                }
                            }
                        }

                        var compactedSubpatterns = [];

                        var currentBasicGraphPatterns = [];
                        var currentFilters = [];

                        for(var i=0; i<subpatterns.length; i++) {
                            if(subpatterns[i].token!='triplespattern' && subpatterns[i].token !=
'filter') {
                                if(currentBasicGraphPatterns.length != 0 || currentFilters.length != 0) {
                                    var triplesContext = [];
                                    for(var j=0; j<currentBasicGraphPatterns.length; j++) {
                                        triplesContext = triplesContext.concat(currentBasicGraphPatterns
[j].triplesContext);
                                    }
                                    if(triplesContext.length > 0) {
                                        compactedSubpatterns.push({token: 'basicgraphpattern',
                                            triplesContext: triplesContext});
                                    }
                                    currentBasicGraphPatterns = [];
                                }
                                compactedSubpatterns.push(subpatterns[i]);
                            } else {
                                if(subpatterns[i].token === 'triplespattern') {
                                    currentBasicGraphPatterns.push(subpatterns[i]);
                                } else {
                                    currentFilters.push(subpatterns[i]);
                                }
                            }
                        }

                        if(currentBasicGraphPatterns.length != 0 || currentFilters.length != 0) {
                            var triplesContext = [];
                            for(var j=0; j<currentBasicGraphPatterns.length; j++) {
                                triplesContext = triplesContext.concat(currentBasicGraphPatterns
[j].triplesContext);
                            }
                            if(triplesContext.length > 0) {
                                compactedSubpatterns.push({token: 'basicgraphpattern',
                                    triplesContext: triplesContext});
                            }
                        }

                        //      if(compactedSubpatterns.length == 1) {
                        //          compactedSubpatterns[0].filters = currentFilters;
                        //          return compactedSubpatterns[0];
                        //      } else  {
                        return { token: 'groupgraphpattern',
                            patterns: compactedSubpatterns,
                            filters: currentFilters }
                        //      }
                    })(result1[0], result1[2])
                        : null;
```

```
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[51] GroupGraphPatternSub");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_TriplesBlock() {
        var cacheKey = 'TriplesBlock@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_TriplesSameSubjectPath();
        if (result3 !== null) {
            var savedPos2 = pos;
            var result6 = [];
            var result10 = parse_WS();
            while (result10 !== null) {
                result6.push(result10);
                var result10 = parse_WS();
            }
            if (result6 !== null) {
                if (input.substr(pos, 1) === ".") {
                    var result7 = ".";
                    pos += 1;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\".\"");
                    }
                }
                if (result7 !== null) {
                    var result9 = parse_TriplesBlock();
                    var result8 = result9 !== null ? result9 : '';
                    if (result8 !== null) {
                        var result5 = [result6, result7, result8];
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
            var result4 = result5 !== null ? result5 : '';
```

```
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(b, bs) {
            var triples = b.triplesContext;
            var toTest = null;
            if(typeof(bs) === 'object') {
                if(bs.length != null) {
                    if(bs[2].triplesContext!=null) {
                        triples = triples.concat(bs[2].triplesContext);
                    }
                }
            }

            return {token:'triplespattern',
                triplesContext: triples}
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[54] TriplesBlock");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_GraphPatternNotTriples() {
        var cacheKey = 'GraphPatternNotTriples@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result6 = parse_GroupOrUnionGraphPattern();
        if (result6 !== null) {
            var result0 = result6;
        } else {
            var result5 = parse_OptionalGraphPattern();
            if (result5 !== null) {
                var result0 = result5;
            } else {
                var result4 = parse_MinusGraphPattern();
                if (result4 !== null) {
                    var result0 = result4;
                } else {
                    var result3 = parse_GraphGraphPattern();
```

```
                    if (result3 !== null) {
                        var result0 = result3;
                    } else {
                        var result2 = parse_ServiceGraphPattern();
                        if (result2 !== null) {
                            var result0 = result2;
                        } else {
                            var result1 = parse_Filter();
                            if (result1 !== null) {
                                var result0 = result1;
                            } else {
                                var result0 = null;;
                            };
                        };
                    };
                };
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[53] GraphPatternNotTriples");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_OptionalGraphPattern() {
        var cacheKey = 'OptionalGraphPattern@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result10 = parse_WS();
        while (result10 !== null) {
            result3.push(result10);
            var result10 = parse_WS();
        }
        if (result3 !== null) {
            if (input.substr(pos, 8) === "OPTIONAL") {
                var result9 = "OPTIONAL";
                pos += 8;
            } else {
                var result9 = null;
                if (reportMatchFailures) {
                    matchFailed("\"OPTIONAL\"");
                }
            }
            if (result9 !== null) {
                var result4 = result9;
            } else {
                if (input.substr(pos, 8) === "optional") {
                    var result8 = "optional";
                    pos += 8;
                } else {
                    var result8 = null;
                    if (reportMatchFailures) {
```

```
                        matchFailed("\"optional\"");
                    }
                }
                if (result8 !== null) {
                    var result4 = result8;
                } else {
                    var result4 = null;;
                };
            }
            if (result4 !== null) {
                var result5 = [];
                var result7 = parse_WS();
                while (result7 !== null) {
                    result5.push(result7);
                    var result7 = parse_WS();
                }
                if (result5 !== null) {
                    var result6 = parse_GroupGraphPattern();
                    if (result6 !== null) {
                        var result1 = [result3, result4, result5, result6];
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(v) {
            return { token: 'optionalgraphpattern',
                value: v }
        })(result1[3])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[54] OptionalGraphPattern");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_GraphGraphPattern() {
        var cacheKey = 'GraphGraphPattern@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }
```

```
                    var savedReportMatchFailures = reportMatchFailures;
                    reportMatchFailures = false;
                    var savedPos0 = pos;
                    var savedPos1 = pos;
                    var result3 = [];
                    var result13 = parse_WS();
                    while (result13 !== null) {
                        result3.push(result13);
                        var result13 = parse_WS();
                    }
                    if (result3 !== null) {
                        if (input.substr(pos, 5) === "GRAPH") {
                            var result12 = "GRAPH";
                            pos += 5;
                        } else {
                            var result12 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"GRAPH\"");
                            }
                        }
                        if (result12 !== null) {
                            var result4 = result12;
                        } else {
                            if (input.substr(pos, 5) === "graph") {
                                var result11 = "graph";
                                pos += 5;
                            } else {
                                var result11 = null;
                                if (reportMatchFailures) {
                                    matchFailed("\"graph\"");
                                }
                            }
                            if (result11 !== null) {
                                var result4 = result11;
                            } else {
                                var result4 = null;;
                            };
                        }
                        if (result4 !== null) {
                            var result5 = [];
                            var result10 = parse_WS();
                            while (result10 !== null) {
                                result5.push(result10);
                                var result10 = parse_WS();
                            }
                            if (result5 !== null) {
                                var result6 = parse_VarOrIRIref();
                                if (result6 !== null) {
                                    var result7 = [];
                                    var result9 = parse_WS();
                                    while (result9 !== null) {
                                        result7.push(result9);
                                        var result9 = parse_WS();
                                    }
                                    if (result7 !== null) {
                                        var result8 = parse_GroupGraphPattern();
                                        if (result8 !== null) {
                                            var result1 = [result3, result4, result5, result6, result7,
result8];
                                        } else {
                                            var result1 = null;
                                            pos = savedPos1;
                                        }
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
```

```
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(g, gg) {
        for(var i=0; i<gg.patterns.length; i++) {
            var quads = []
            var ts = gg.patterns[i];
            for(var j=0; j<ts.triplesContext.length; j++) {
                var triple = ts.triplesContext[j]
                triple.graph = g;
            }
        }

        gg.token = 'groupgraphpattern'
        return gg;
    })(result1[3], result1[5])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[55] GraphGraphPattern");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_ServiceGraphPattern() {
    var cacheKey = 'ServiceGraphPattern@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    var savedPos1 = pos;
    if (input.substr(pos, 7) === "SERVICE") {
        var result3 = "SERVICE";
        pos += 7;
    } else {
        var result3 = null;
```

```
            if (reportMatchFailures) {
                matchFailed("\"SERVICE\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_VarOrIRIref();
            if (result4 !== null) {
                var result5 = parse_GroupGraphPattern();
                if (result5 !== null) {
                    var result1 = [result3, result4, result5];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(v, ts) {
            return {token: 'servicegraphpattern',
                status: 'todo',
                value: [v,ts] }
        })(result1[1], result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[56] ServiceGraphPattern");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_MinusGraphPattern() {
        var cacheKey = 'MinusGraphPattern@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 5) === "MINUS") {
            var result3 = "MINUS";
            pos += 5;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"MINUS\"");
            }
```

```
        }
        if (result3 !== null) {
            var result4 = parse_GroupGraphPattern();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(ts) {
            return {token: 'minusgraphpattern',
                status: 'todo',
                value: ts}
        })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[57] MinusGraphPattern");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_GroupOrUnionGraphPattern() {
        var cacheKey = 'GroupOrUnionGraphPattern@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_GroupGraphPattern();
        if (result3 !== null) {
            var result4 = [];
            var savedPos2 = pos;
            var result6 = [];
            var result13 = parse_WS();
            while (result13 !== null) {
                result6.push(result13);
                var result13 = parse_WS();
            }
            if (result6 !== null) {
                if (input.substr(pos, 5) === "UNION") {
                    var result12 = "UNION";
                    pos += 5;
                } else {
                    var result12 = null;
                    if (reportMatchFailures) {
```

```
                    matchFailed("\"UNION\"");
                }
            }
            if (result12 !== null) {
                var result7 = result12;
            } else {
                if (input.substr(pos, 5) === "union") {
                    var result11 = "union";
                    pos += 5;
                } else {
                    var result11 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"union\"");
                    }
                }
                if (result11 !== null) {
                    var result7 = result11;
                } else {
                    var result7 = null;;
                };
            }
            if (result7 !== null) {
                var result8 = [];
                var result10 = parse_WS();
                while (result10 !== null) {
                    result8.push(result10);
                    var result10 = parse_WS();
                }
                if (result8 !== null) {
                    var result9 = parse_GroupGraphPattern();
                    if (result9 !== null) {
                        var result5 = [result6, result7, result8, result9];
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
        } else {
            var result5 = null;
            pos = savedPos2;
        }
        while (result5 !== null) {
            result4.push(result5);
            var savedPos2 = pos;
            var result6 = [];
            var result13 = parse_WS();
            while (result13 !== null) {
                result6.push(result13);
                var result13 = parse_WS();
            }
            if (result6 !== null) {
                if (input.substr(pos, 5) === "UNION") {
                    var result12 = "UNION";
                    pos += 5;
                } else {
                    var result12 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"UNION\"");
                    }
                }
```

```
                    if (result12 !== null) {
                        var result7 = result12;
                    } else {
                        if (input.substr(pos, 5) === "union") {
                            var result11 = "union";
                            pos += 5;
                        } else {
                            var result11 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"union\"");
                            }
                        }
                        if (result11 !== null) {
                            var result7 = result11;
                        } else {
                            var result7 = null;;
                        };
                    }
                    if (result7 !== null) {
                        var result8 = [];
                        var result10 = parse_WS();
                        while (result10 !== null) {
                            result8.push(result10);
                            var result10 = parse_WS();
                        }
                        if (result8 !== null) {
                            var result9 = parse_GroupGraphPattern();
                            if (result9 !== null) {
                                var result5 = [result6, result7, result8, result9];
                            } else {
                                var result5 = null;
                                pos = savedPos2;
                            }
                        } else {
                            var result5 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            }
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(a, b) {
            if(b.length === 0) {
                return a;
            } else {

                var lastToken = {token: 'graphunionpattern',
                    value: [a]};

                for(var i=0; i<b.length; i++) {
                    if(i==b.length-1) {
```

```
                    lastToken.value.push(b[i][3]);
                } else {
                    lastToken.value.push(b[i][3]);
                    var newToken = {token: 'graphunionpattern',
                        value: [lastToken]}

                    lastToken = newToken;
                }
            }

            return lastToken;

        }
    })(result1[0], result1[1])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[58] GroupOrUnionGraphPattern");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_Filter() {
    var cacheKey = 'Filter@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    var savedPos1 = pos;
    var result3 = [];
    var result10 = parse_WS();
    while (result10 !== null) {
        result3.push(result10);
        var result10 = parse_WS();
    }
    if (result3 !== null) {
        if (input.substr(pos, 6) === "FILTER") {
            var result9 = "FILTER";
            pos += 6;
        } else {
            var result9 = null;
            if (reportMatchFailures) {
                matchFailed("\"FILTER\"");
            }
        }
        if (result9 !== null) {
            var result4 = result9;
        } else {
            if (input.substr(pos, 6) === "filter") {
                var result8 = "filter";
                pos += 6;
```

```
                } else {
                    var result8 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"filter\"");
                    }
                }
                if (result8 !== null) {
                    var result4 = result8;
                } else {
                    var result4 = null;;
                };
            }
            if (result4 !== null) {
                var result5 = [];
                var result7 = parse_WS();
                while (result7 !== null) {
                    result5.push(result7);
                    var result7 = parse_WS();
                }
                if (result5 !== null) {
                    var result6 = parse_Constraint();
                    if (result6 !== null) {
                        var result1 = [result3, result4, result5, result6];
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(c) {
            return {token: 'filter',
                value: c}
        })(result1[3])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[59] Filter");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Constraint() {
        var cacheKey = 'Constraint@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
```

```
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result3 = parse_BrackettedExpression();
        if (result3 !== null) {
            var result0 = result3;
        } else {
            var result2 = parse_BuiltInCall();
            if (result2 !== null) {
                var result0 = result2;
            } else {
                var result1 = parse_FunctionCall();
                if (result1 !== null) {
                    var result0 = result1;
                } else {
                    var result0 = null;;
                };
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[60] Constraint");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_FunctionCall() {
        var cacheKey = 'FunctionCall@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_IRIref();
        if (result3 !== null) {
            var result4 = parse_ArgList();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(i, args) {
            var fcall = {};
            fcall.token = "expression";
            fcall.expressionType = 'irireforfunction'
            fcall.iriref = i;
            fcall.args = args.value;
```

```
            return fcall;
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[61] FunctionCall");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ArgList() {
        var cacheKey = 'ArgList@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos3 = pos;
        var result16 = parse_NIL();
        var result17 = result16 !== null
            ? (function() {
            var args = {};
            args.token = 'args';
            args.value = [];
            return args;
        })()
            : null;
        if (result17 !== null) {
            var result15 = result17;
        } else {
            var result15 = null;
            pos = savedPos3;
        }
        if (result15 !== null) {
            var result0 = result15;
        } else {
            var savedPos0 = pos;
            var savedPos1 = pos;
            if (input.substr(pos, 1) === "(") {
                var result4 = "(";
                pos += 1;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("\"(\"");
                }
            }
            if (result4 !== null) {
                if (input.substr(pos, 8) === "DISTINCT") {
                    var result14 = "DISTINCT";
                    pos += 8;
                } else {
                    var result14 = null;
```

```
        if (reportMatchFailures) {
            matchFailed("\"DISTINCT\"");
        }
    }
    if (result14 !== null) {
        var result12 = result14;
    } else {
        if (input.substr(pos, 8) === "distinct") {
            var result13 = "distinct";
            pos += 8;
        } else {
            var result13 = null;
            if (reportMatchFailures) {
                matchFailed("\"distinct\"");
            }
        }
        if (result13 !== null) {
            var result12 = result13;
        } else {
            var result12 = null;;
        };
    }
    var result5 = result12 !== null ? result12 : '';
    if (result5 !== null) {
        var result6 = parse_ConditionalOrExpression();
        if (result6 !== null) {
            var result7 = [];
            var savedPos2 = pos;
            if (input.substr(pos, 1) === ",") {
                var result10 = ",";
                pos += 1;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("\",\"");
                }
            }
            if (result10 !== null) {
                var result11 = parse_ConditionalOrExpression();
                if (result11 !== null) {
                    var result9 = [result10, result11];
                } else {
                    var result9 = null;
                    pos = savedPos2;
                }
            } else {
                var result9 = null;
                pos = savedPos2;
            }
            while (result9 !== null) {
                result7.push(result9);
                var savedPos2 = pos;
                if (input.substr(pos, 1) === ",") {
                    var result10 = ",";
                    pos += 1;
                } else {
                    var result10 = null;
                    if (reportMatchFailures) {
                        matchFailed("\",\"");
                    }
                }
                if (result10 !== null) {
                    var result11 = parse_ConditionalOrExpression();
                    if (result11 !== null) {
                        var result9 = [result10, result11];
                    } else {
                        var result9 = null;
```

```
                            pos = savedPos2;
                        }
                    } else {
                        var result9 = null;
                        pos = savedPos2;
                    }
                }
                if (result7 !== null) {
                    if (input.substr(pos, 1) === ")") {
                        var result8 = ")";
                        pos += 1;
                    } else {
                        var result8 = null;
                        if (reportMatchFailures) {
                            matchFailed("\")\"");
                        }
                    }
                    if (result8 !== null) {
                        var result2 = [result4, result5, result6, result7, result8];
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result2 = null;
                    pos = savedPos1;
                }
            } else {
                var result2 = null;
                pos = savedPos1;
            }
        } else {
            var result2 = null;
            pos = savedPos1;
        }
    } else {
        var result2 = null;
        pos = savedPos1;
    }
    var result3 = result2 !== null
        ? (function(d, e, es) {
        var cleanEx = [];

        for(var i=0; i<es.length; i++) {
            cleanEx.push(es[i][1]);
        }
        var args = {};
        args.token = 'args';
        args.value = [e].concat(cleanEx);

        if(d!=null && d.toUpperCase()==="DISTINCT") {
            args.distinct = true;
        } else {
            args.distinct = false;
        }

        return args;
    })(result2[1], result2[2], result2[3])
        : null;
    if (result3 !== null) {
        var result1 = result3;
    } else {
        var result1 = null;
        pos = savedPos0;
    }
    if (result1 !== null) {
        var result0 = result1;
```

```
            } else {
                var result0 = null;;
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[62] ArgList");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ExpressionList() {
        var cacheKey = 'ExpressionList@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos3 = pos;
        var result12 = parse_NIL();
        var result13 = result12 !== null
            ? (function() {
            var args = {};
            args.token = 'args';
            args.value = [];
            return args;
        })()
            : null;
        if (result13 !== null) {
            var result11 = result13;
        } else {
            var result11 = null;
            pos = savedPos3;
        }
        if (result11 !== null) {
            var result0 = result11;
        } else {
            var savedPos0 = pos;
            var savedPos1 = pos;
            if (input.substr(pos, 1) === "(") {
                var result4 = "(";
                pos += 1;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("\"(\"");
                }
            }
            if (result4 !== null) {
                var result5 = parse_ConditionalOrExpression();
                if (result5 !== null) {
                    var result6 = [];
                    var savedPos2 = pos;
                    if (input.substr(pos, 1) === ",") {
                        var result9 = ",";
                        pos += 1;
                    } else {
                        var result9 = null;
                        if (reportMatchFailures) {
```

```
                        matchFailed("\",\"");
                    }
                }
                if (result9 !== null) {
                    var result10 = parse_ConditionalOrExpression();
                    if (result10 !== null) {
                        var result8 = [result9, result10];
                    } else {
                        var result8 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result8 = null;
                    pos = savedPos2;
                }
                while (result8 !== null) {
                    result6.push(result8);
                    var savedPos2 = pos;
                    if (input.substr(pos, 1) === ",") {
                        var result9 = ",";
                        pos += 1;
                    } else {
                        var result9 = null;
                        if (reportMatchFailures) {
                            matchFailed("\",\"");
                        }
                    }
                    if (result9 !== null) {
                        var result10 = parse_ConditionalOrExpression();
                        if (result10 !== null) {
                            var result8 = [result9, result10];
                        } else {
                            var result8 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result8 = null;
                        pos = savedPos2;
                    }
                }
                if (result6 !== null) {
                    if (input.substr(pos, 1) === ")") {
                        var result7 = ")";
                        pos += 1;
                    } else {
                        var result7 = null;
                        if (reportMatchFailures) {
                            matchFailed("\")\"");
                        }
                    }
                    if (result7 !== null) {
                        var result2 = [result4, result5, result6, result7];
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result2 = null;
                    pos = savedPos1;
                }
            } else {
                var result2 = null;
                pos = savedPos1;
            }
        } else {
            var result2 = null;
            pos = savedPos1;
```

```
            }
            var result3 = result2 !== null
                ? (function(e, es) {
                var cleanEx = [];

                for(var i=0; i<es.length; i++) {
                    cleanEx.push(es[i][1]);
                }
                var args = {};
                args.token = 'args';
                args.value = [e].concat(cleanEx);

                return args;
            })(result2[1], result2[2])
                : null;
            if (result3 !== null) {
                var result1 = result3;
            } else {
                var result1 = null;
                pos = savedPos0;
            }
            if (result1 !== null) {
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[63] ExpressionList");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ConstructTemplate() {
        var cacheKey = 'ConstructTemplate@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "{") {
            var result3 = "{";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"{\"");
            }
        }
        if (result3 !== null) {
            var result4 = [];
            var result10 = parse_WS();
            while (result10 !== null) {
                result4.push(result10);
                var result10 = parse_WS();
            }
```

```
            if (result4 !== null) {
                var result9 = parse_ConstructTriples();
                var result5 = result9 !== null ? result9 : '';
                if (result5 !== null) {
                    var result6 = [];
                    var result8 = parse_WS();
                    while (result8 !== null) {
                        result6.push(result8);
                        var result8 = parse_WS();
                    }
                    if (result6 !== null) {
                        if (input.substr(pos, 1) === "}") {
                            var result7 = "}";
                            pos += 1;
                        } else {
                            var result7 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"}\"");
                            }
                        }
                        if (result7 !== null) {
                            var result1 = [result3, result4, result5, result6, result7];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(ts) {
            return ts;
        })(result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[64] ConstructTemplate");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ConstructTriples() {
        var cacheKey = 'ConstructTriples@' + pos;
```

```
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_TriplesSameSubject();
        if (result3 !== null) {
            var savedPos2 = pos;
            var result6 = [];
            var result12 = parse_WS();
            while (result12 !== null) {
                result6.push(result12);
                var result12 = parse_WS();
            }
            if (result6 !== null) {
                if (input.substr(pos, 1) === ".") {
                    var result7 = ".";
                    pos += 1;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\".\"");
                    }
                }
                if (result7 !== null) {
                    var result8 = [];
                    var result11 = parse_WS();
                    while (result11 !== null) {
                        result8.push(result11);
                        var result11 = parse_WS();
                    }
                    if (result8 !== null) {
                        var result10 = parse_ConstructTriples();
                        var result9 = result10 !== null ? result10 : '';
                        if (result9 !== null) {
                            var result5 = [result6, result7, result8, result9];
                        } else {
                            var result5 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
            var result4 = result5 !== null ? result5 : '';
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
```

```
        }
        var result2 = result1 !== null
            ? (function(b, bs) {
            var triples = b.triplesContext;
            var toTest = null;
            if(typeof(bs) === 'object') {
                if(bs.length != null) {
                    if(bs[3].triplesContext!=null) {
                        triples = triples.concat(bs[3].triplesContext);
                    }
                }
            }

            return {token:'triplestemplate',
                triplesContext: triples}
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[65] ConstructTriples");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_TriplesSameSubject() {
        var cacheKey = 'TriplesSameSubject@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos2 = pos;
        var savedPos3 = pos;
        var result13 = [];
        var result18 = parse_WS();
        while (result18 !== null) {
            result13.push(result18);
            var result18 = parse_WS();
        }
        if (result13 !== null) {
            var result14 = parse_VarOrTerm();
            if (result14 !== null) {
                var result15 = [];
                var result17 = parse_WS();
                while (result17 !== null) {
                    result15.push(result17);
                    var result17 = parse_WS();
                }
                if (result15 !== null) {
                    var result16 = parse_PropertyListNotEmpty();
                    if (result16 !== null) {
                        var result11 = [result13, result14, result15, result16];
                    } else {
```

```
                                var result11 = null;
                                pos = savedPos3;
                            }
                        } else {
                            var result11 = null;
                            pos = savedPos3;
                        }
                    } else {
                        var result11 = null;
                        pos = savedPos3;
                    }
                } else {
                    var result11 = null;
                    pos = savedPos3;
                }
                var result12 = result11 !== null
                    ? (function(s, pairs) {
                    var triplesContext = pairs.triplesContext;
                    var subject = s;
                    if(pairs.pairs) {
                        for(var i=0; i< pairs.pairs.length; i++) {
                            var pair = pairs.pairs[i];
                            var triple = null;
                            if(pair[1].length != null)
                                pair[1] = pair[1][0]
                            if(subject.token && subject.token==='triplesnodecollection') {
                                triple = {subject: subject.chainSubject[0], predicate: pair[0],
object: pair[1]}
                                triplesContext.push(triple);
                                triplesContext = triplesContext.concat(subject.triplesContext);
                            } else {
                                triple = {subject: subject, predicate: pair[0], object: pair[1]}
                                triplesContext.push(triple);
                            }
                        }
                    }

                    var token = {};
                    token.token = "triplessamesubject";
                    token.triplesContext = triplesContext;
                    token.chainSubject = subject;

                    return token;
                })(result11[1], result11[3])
                    : null;
                if (result12 !== null) {
                    var result10 = result12;
                } else {
                    var result10 = null;
                    pos = savedPos2;
                }
                if (result10 !== null) {
                    var result0 = result10;
                } else {
                    var savedPos0 = pos;
                    var savedPos1 = pos;
                    var result4 = [];
                    var result9 = parse_WS();
                    while (result9 !== null) {
                        result4.push(result9);
                        var result9 = parse_WS();
                    }
                    if (result4 !== null) {
                        var result5 = parse_TriplesNode();
                        if (result5 !== null) {
                            var result6 = [];
                            var result8 = parse_WS();
```

```
                        while (result8 !== null) {
                            result6.push(result8);
                            var result8 = parse_WS();
                        }
                        if (result6 !== null) {
                            var result7 = parse_PropertyList();
                            if (result7 !== null) {
                                var result2 = [result4, result5, result6, result7];
                            } else {
                                var result2 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result2 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result2 = null;
                    pos = savedPos1;
                }
                var result3 = result2 !== null
                    ? (function(tn, pairs) {
                    var triplesContext = tn.triplesContext;
                    var subject = tn.chainSubject;

                    if(pairs.pairs) {
                        for(var i=0; i< pairs.pairs.length; i++) {
                            var pair = pairs.pairs[i];
                            if(pair[1].length != null)
                                pair[1] = pair[1][0]

                            if(tn.token === "triplesnodecollection") {
                                for(var j=0; j<subject.length; j++) {
                                    var subj = subject[j];
                                    if(subj.triplesContext != null) {
                                        var triple = {subject: subj.chainSubject, predicate: pair
[0], object: pair[1]}
                                        triplesContext.concat(subj.triplesContext);
                                    } else {
                                        var triple = {subject: subject[j], predicate: pair[0],
object: pair[1]}
                                        triplesContext.push(triple);
                                    }
                                }
                            } else {
                                var triple = {subject: subject, predicate: pair[0], object: pair
[1]}
                                triplesContext.push(triple);
                            }
                        }
                    }

                    var token = {};
                    token.token = "triplessamesubject";
                    token.triplesContext = triplesContext;
                    token.chainSubject = subject;

                    return token;
                })(result2[1], result2[3])
                    : null;
                if (result3 !== null) {
                    var result1 = result3;
                } else {
```

```
                var result1 = null;
                pos = savedPos0;
            }
            if (result1 !== null) {
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[66] TriplesSameSubject");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PropertyListNotEmpty() {
        var cacheKey = 'PropertyListNotEmpty@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_Verb();
        if (result3 !== null) {
            var result4 = [];
            var result19 = parse_WS();
            while (result19 !== null) {
                result4.push(result19);
                var result19 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = parse_ObjectList();
                if (result5 !== null) {
                    var result6 = [];
                    var savedPos2 = pos;
                    var result8 = [];
                    var result18 = parse_WS();
                    while (result18 !== null) {
                        result8.push(result18);
                        var result18 = parse_WS();
                    }
                    if (result8 !== null) {
                        if (input.substr(pos, 1) === ";") {
                            var result9 = ";";
                            pos += 1;
                        } else {
                            var result9 = null;
                            if (reportMatchFailures) {
                                matchFailed("\";\"");
                            }
                        }
                        if (result9 !== null) {
                            var result10 = [];
                            var result17 = parse_WS();
                            while (result17 !== null) {
                                result10.push(result17);
```

```
                    var result17 = parse_WS();
                }
                if (result10 !== null) {
                    var savedPos3 = pos;
                    var result13 = parse_Verb();
                    if (result13 !== null) {
                        var result14 = [];
                        var result16 = parse_WS();
                        while (result16 !== null) {
                            result14.push(result16);
                            var result16 = parse_WS();
                        }
                        if (result14 !== null) {
                            var result15 = parse_ObjectList();
                            if (result15 !== null) {
                                var result12 = [result13, result14, result15];
                            } else {
                                var result12 = null;
                                pos = savedPos3;
                            }
                        } else {
                            var result12 = null;
                            pos = savedPos3;
                        }
                    } else {
                        var result12 = null;
                        pos = savedPos3;
                    }
                    var result11 = result12 !== null ? result12 : '';
                    if (result11 !== null) {
                        var result7 = [result8, result9, result10, result11];
                    } else {
                        var result7 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result7 = null;
                    pos = savedPos2;
                }
            } else {
                var result7 = null;
                pos = savedPos2;
            }
        } else {
            var result7 = null;
            pos = savedPos2;
        }
        while (result7 !== null) {
            result6.push(result7);
            var savedPos2 = pos;
            var result8 = [];
            var result18 = parse_WS();
            while (result18 !== null) {
                result8.push(result18);
                var result18 = parse_WS();
            }
            if (result8 !== null) {
                if (input.substr(pos, 1) === ";") {
                    var result9 = ";";
                    pos += 1;
                } else {
                    var result9 = null;
                    if (reportMatchFailures) {
                        matchFailed("\";\"");
                    }
                }
                if (result9 !== null) {
```

```
                                var result10 = [];
                                var result17 = parse_WS();
                                while (result17 !== null) {
                                    result10.push(result17);
                                    var result17 = parse_WS();
                                }
                                if (result10 !== null) {
                                    var savedPos3 = pos;
                                    var result13 = parse_Verb();
                                    if (result13 !== null) {
                                        var result14 = [];
                                        var result16 = parse_WS();
                                        while (result16 !== null) {
                                            result14.push(result16);
                                            var result16 = parse_WS();
                                        }
                                        if (result14 !== null) {
                                            var result15 = parse_ObjectList();
                                            if (result15 !== null) {
                                                var result12 = [result13, result14, result15];
                                            } else {
                                                var result12 = null;
                                                pos = savedPos3;
                                            }
                                        } else {
                                            var result12 = null;
                                            pos = savedPos3;
                                        }
                                    } else {
                                        var result12 = null;
                                        pos = savedPos3;
                                    }
                                    var result11 = result12 !== null ? result12 : '';
                                    if (result11 !== null) {
                                        var result7 = [result8, result9, result10, result11];
                                    } else {
                                        var result7 = null;
                                        pos = savedPos2;
                                    }
                                } else {
                                    var result7 = null;
                                    pos = savedPos2;
                                }
                            } else {
                                var result7 = null;
                                pos = savedPos2;
                            }
                        } else {
                            var result7 = null;
                            pos = savedPos2;
                        }
                        if (result6 !== null) {
                            var result1 = [result3, result4, result5, result6];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
```

```
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function(v, ol, rest) {
                    var token = {}
                    token.token = 'propertylist';
                    var triplesContext = [];
                    var pairs = [];
                    var test = [];

                    for( var i=0; i<ol.length; i++) {

                        if(ol[i].triplesContext != null) {
                            triplesContext = triplesContext.concat(ol[i].triplesContext);
                            if(ol[i].token==='triplesnodecollection' && ol[i].chainSubject.length !=
null) {
                                pairs.push([v, ol[i].chainSubject[0]]);
                            } else {
                                pairs.push([v, ol[i].chainSubject]);
                            }

                        } else {
                            pairs.push([v, ol[i]])
                        }

                    }


                    for(var i=0; i<rest.length; i++) {
                        var tok = rest[i][3];
                        var newVerb  = tok[0];
                        var newObjsList = tok[2] || [];

                        for(var j=0; j<newObjsList.length; j++) {
                            if(newObjsList[j].triplesContext != null) {
                                triplesContext = triplesContext.concat(newObjsList[j].triplesContext);
                                pairs.push([newVerb, newObjsList[j].chainSubject]);
                            } else {
                                pairs.push([newVerb, newObjsList[j]])
                            }
                        }
                    }

                    token.pairs = pairs;
                    token.triplesContext = triplesContext;

                    return token;

                })(result1[0], result1[2], result1[3])
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[67] PropertyListNotEmpty");
                }

                cache[cacheKey] = {
                    nextPos: pos,
                    result:  result0
                };
                return result0;
```

```
    }

    function parse_PropertyList() {
        var cacheKey = 'PropertyList@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result1 = parse_PropertyListNotEmpty();
        var result0 = result1 !== null ? result1 : '';
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[68] PropertyList");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ObjectList() {
        var cacheKey = 'ObjectList@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_GraphNode();
        if (result3 !== null) {
            var result4 = [];
            var result11 = parse_WS();
            while (result11 !== null) {
                result4.push(result11);
                var result11 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = [];
                var savedPos2 = pos;
                if (input.substr(pos, 1) === ",") {
                    var result7 = ",";
                    pos += 1;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\",\"");
                    }
                }
                if (result7 !== null) {
                    var result8 = [];
                    var result10 = parse_WS();
                    while (result10 !== null) {
                        result8.push(result10);
                        var result10 = parse_WS();
                    }
                    if (result8 !== null) {
                        var result9 = parse_GraphNode();
```

```
                    if (result9 !== null) {
                        var result6 = [result7, result8, result9];
                    } else {
                        var result6 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result6 = null;
                    pos = savedPos2;
                }
            } else {
                var result6 = null;
                pos = savedPos2;
            }
            while (result6 !== null) {
                result5.push(result6);
                var savedPos2 = pos;
                if (input.substr(pos, 1) === ",") {
                    var result7 = ",";
                    pos += 1;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\",\"");
                    }
                }
                if (result7 !== null) {
                    var result8 = [];
                    var result10 = parse_WS();
                    while (result10 !== null) {
                        result8.push(result10);
                        var result10 = parse_WS();
                    }
                    if (result8 !== null) {
                        var result9 = parse_GraphNode();
                        if (result9 !== null) {
                            var result6 = [result7, result8, result9];
                        } else {
                            var result6 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result6 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result6 = null;
                    pos = savedPos2;
                }
            }
            if (result5 !== null) {
                var result1 = [result3, result4, result5];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(obj, objs) {
```

```
                    var toReturn = [];

                    toReturn.push(obj);

                    for(var i=0; i<objs.length; i++) {
                        for(var j=0; j<objs[i].length; j++) {
                            if(typeof(objs[i][j])=="object" && objs[i][j].token != null) {
                                toReturn.push(objs[i][j]);
                            }
                        }
                    }

                    return toReturn;
                })(result1[0], result1[2])
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[69] ObjectList");
                }

                cache[cacheKey] = {
                    nextPos: pos,
                    result:  result0
                };
                return result0;
            }

            function parse_Verb() {
                var cacheKey = 'Verb@' + pos;
                var cachedResult = cache[cacheKey];
                if (cachedResult) {
                    pos = cachedResult.nextPos;
                    return cachedResult.result;
                }

                var savedReportMatchFailures = reportMatchFailures;
                reportMatchFailures = false;
                var result4 = parse_VarOrIRIref();
                if (result4 !== null) {
                    var result0 = result4;
                } else {
                    var savedPos0 = pos;
                    if (input.substr(pos, 1) === "a") {
                        var result2 = "a";
                        pos += 1;
                    } else {
                        var result2 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"a\"");
                        }
                    }
                    var result3 = result2 !== null
                        ? (function() {
                        return{token: 'uri', prefix:null, suffix:null, value:"http://
www.w3.org/1999/02/22-rdf-syntax-ns#type"}
                    })()
                        : null;
                    if (result3 !== null) {
                        var result1 = result3;
                    } else {
                        var result1 = null;
```

```
                pos = savedPos0;
            }
            if (result1 !== null) {
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[71] Verb");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_TriplesSameSubjectPath() {
        var cacheKey = 'TriplesSameSubjectPath@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos2 = pos;
        var savedPos3 = pos;
        var result13 = [];
        var result18 = parse_WS();
        while (result18 !== null) {
            result13.push(result18);
            var result18 = parse_WS();
        }
        if (result13 !== null) {
            var result14 = parse_VarOrTerm();
            if (result14 !== null) {
                var result15 = [];
                var result17 = parse_WS();
                while (result17 !== null) {
                    result15.push(result17);
                    var result17 = parse_WS();
                }
                if (result15 !== null) {
                    var result16 = parse_PropertyListNotEmptyPath();
                    if (result16 !== null) {
                        var result11 = [result13, result14, result15, result16];
                    } else {
                        var result11 = null;
                        pos = savedPos3;
                    }
                } else {
                    var result11 = null;
                    pos = savedPos3;
                }
            } else {
                var result11 = null;
                pos = savedPos3;
            }
        } else {
            var result11 = null;
            pos = savedPos3;
        }
```

```
                    var result12 = result11 !== null
                        ? (function(s, pairs) {
                        var triplesContext = pairs.triplesContext;
                        var subject = s;
                        if(pairs.pairs) {
                            for(var i=0; i< pairs.pairs.length; i++) {
                                var pair = pairs.pairs[i];
                                var triple = null;
                                if(pair[1].length != null)
                                    pair[1] = pair[1][0]
                                if(subject.token && subject.token==='triplesnodecollection') {
                                    triple = {subject: subject.chainSubject[0], predicate: pair[0],
object: pair[1]}
                                    triplesContext.push(triple);
                                    triplesContext = triplesContext.concat(subject.triplesContext);
                                } else {
                                    triple = {subject: subject, predicate: pair[0], object: pair[1]}
                                    triplesContext.push(triple);
                                }
                            }
                        }

                        var token = {};
                        token.token = "triplessamesubject";
                        token.triplesContext = triplesContext;
                        token.chainSubject = subject;

                        return token;
                    })(result11[1], result11[3])
                        : null;
                    if (result12 !== null) {
                        var result10 = result12;
                    } else {
                        var result10 = null;
                        pos = savedPos2;
                    }
                    if (result10 !== null) {
                        var result0 = result10;
                    } else {
                        var savedPos0 = pos;
                        var savedPos1 = pos;
                        var result4 = [];
                        var result9 = parse_WS();
                        while (result9 !== null) {
                            result4.push(result9);
                            var result9 = parse_WS();
                        }
                        if (result4 !== null) {
                            var result5 = parse_TriplesNode();
                            if (result5 !== null) {
                                var result6 = [];
                                var result8 = parse_WS();
                                while (result8 !== null) {
                                    result6.push(result8);
                                    var result8 = parse_WS();
                                }
                                if (result6 !== null) {
                                    var result7 = parse_PropertyListPath();
                                    if (result7 !== null) {
                                        var result2 = [result4, result5, result6, result7];
                                    } else {
                                        var result2 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result2 = null;
                                    pos = savedPos1;
```

```
                            }
                        } else {
                            var result2 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                    var result3 = result2 !== null
                        ? (function(tn, pairs) {
                        var triplesContext = tn.triplesContext;
                        var subject = tn.chainSubject;

                        if(pairs.pairs) {
                            for(var i=0; i< pairs.pairs.length; i++) {
                                var pair = pairs.pairs[i];
                                if(pair[1].length != null)
                                    pair[1] = pair[1][0]

                                if(tn.token === "triplesnodecollection") {
                                    for(var j=0; j<subject.length; j++) {
                                        var subj = subject[j];
                                        if(subj.triplesContext != null) {
                                            var triple = {subject: subj.chainSubject, predicate: pair
[0], object: pair[1]}

                                            triplesContext.concat(subj.triplesContext);
                                        } else {
                                            var triple = {subject: subject[j], predicate: pair[0],
object: pair[1]}

                                            triplesContext.push(triple);
                                        }
                                    }
                                } else {
                                    var triple = {subject: subject, predicate: pair[0], object: pair
[1]}

                                    triplesContext.push(triple);
                                }
                            }
                        }

                        var token = {};
                        token.token = "triplessamesubject";
                        token.triplesContext = triplesContext;
                        token.chainSubject = subject;

                        return token;

                    })(result2[1], result2[3])
                        : null;
                    if (result3 !== null) {
                        var result1 = result3;
                    } else {
                        var result1 = null;
                        pos = savedPos0;
                    }
                    if (result1 !== null) {
                        var result0 = result1;
                    } else {
                        var result0 = null;;
                    };
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[72] TriplesSameSubjectPath");
                }
```

```
        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PropertyListNotEmptyPath() {
        var cacheKey = 'PropertyListNotEmptyPath@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result21 = parse_VerbPath();
        if (result21 !== null) {
            var result3 = result21;
        } else {
            var result20 = parse_Var();
            if (result20 !== null) {
                var result3 = result20;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result19 = parse_WS();
            while (result19 !== null) {
                result4.push(result19);
                var result19 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = parse_ObjectList();
                if (result5 !== null) {
                    var result6 = [];
                    var savedPos2 = pos;
                    var result8 = [];
                    var result18 = parse_WS();
                    while (result18 !== null) {
                        result8.push(result18);
                        var result18 = parse_WS();
                    }
                    if (result8 !== null) {
                        if (input.substr(pos, 1) === ";") {
                            var result9 = ";";
                            pos += 1;
                        } else {
                            var result9 = null;
                            if (reportMatchFailures) {
                                matchFailed("\";\"");
                            }
                        }
                        if (result9 !== null) {
                            var result10 = [];
                            var result17 = parse_WS();
                            while (result17 !== null) {
                                result10.push(result17);
                                var result17 = parse_WS();
                            }
                            if (result10 !== null) {
                                var savedPos3 = pos;
```

```
                    var result16 = parse_VerbPath();
                    if (result16 !== null) {
                        var result13 = result16;
                    } else {
                        var result15 = parse_Var();
                        if (result15 !== null) {
                            var result13 = result15;
                        } else {
                            var result13 = null;;
                        };
                    }
                    if (result13 !== null) {
                        var result14 = parse_ObjectList();
                        if (result14 !== null) {
                            var result12 = [result13, result14];
                        } else {
                            var result12 = null;
                            pos = savedPos3;
                        }
                    } else {
                        var result12 = null;
                        pos = savedPos3;
                    }
                    var result11 = result12 !== null ? result12 : '';
                    if (result11 !== null) {
                        var result7 = [result8, result9, result10, result11];
                    } else {
                        var result7 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result7 = null;
                    pos = savedPos2;
                }
            } else {
                var result7 = null;
                pos = savedPos2;
            }
        } else {
            var result7 = null;
            pos = savedPos2;
        }
        while (result7 !== null) {
            result6.push(result7);
            var savedPos2 = pos;
            var result8 = [];
            var result18 = parse_WS();
            while (result18 !== null) {
                result8.push(result18);
                var result18 = parse_WS();
            }
            if (result8 !== null) {
                if (input.substr(pos, 1) === ";") {
                    var result9 = ";";
                    pos += 1;
                } else {
                    var result9 = null;
                    if (reportMatchFailures) {
                        matchFailed("\";\"");
                    }
                }
                if (result9 !== null) {
                    var result10 = [];
                    var result17 = parse_WS();
                    while (result17 !== null) {
                        result10.push(result17);
                        var result17 = parse_WS();
```

```
                                }
                                if (result10 !== null) {
                                    var savedPos3 = pos;
                                    var result16 = parse_VerbPath();
                                    if (result16 !== null) {
                                        var result13 = result16;
                                    } else {
                                        var result15 = parse_Var();
                                        if (result15 !== null) {
                                            var result13 = result15;
                                        } else {
                                            var result13 = null;;
                                        };
                                    }
                                    if (result13 !== null) {
                                        var result14 = parse_ObjectList();
                                        if (result14 !== null) {
                                            var result12 = [result13, result14];
                                        } else {
                                            var result12 = null;
                                            pos = savedPos3;
                                        }
                                    } else {
                                        var result12 = null;
                                        pos = savedPos3;
                                    }
                                    var result11 = result12 !== null ? result12 : '';
                                    if (result11 !== null) {
                                        var result7 = [result8, result9, result10, result11];
                                    } else {
                                        var result7 = null;
                                        pos = savedPos2;
                                    }
                                } else {
                                    var result7 = null;
                                    pos = savedPos2;
                                }
                            } else {
                                var result7 = null;
                                pos = savedPos2;
                            }
                        } else {
                            var result7 = null;
                            pos = savedPos2;
                        }
                    }
                    if (result6 !== null) {
                        var result1 = [result3, result4, result5, result6];
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(v, ol, rest) {
            token = {}
```

```
                        token.token = 'propertylist';
                        var triplesContext = [];
                        var pairs = [];
                        var test = [];

                        for( var i=0; i<ol.length; i++) {

                            if(ol[i].triplesContext != null) {
                                triplesContext = triplesContext.concat(ol[i].triplesContext);
                                if(ol[i].token==='triplesnodecollection' && ol[i].chainSubject.length !=
null) {
                                    pairs.push([v, ol[i].chainSubject[0]]);
                                } else {
                                    pairs.push([v, ol[i].chainSubject]);
                                }

                            } else {
                                pairs.push([v, ol[i]])
                            }

                        }


                        for(var i=0; i<rest.length; i++) {
                            var tok = rest[i][3];
                            var newVerb  = tok[0];
                            var newObjsList = tok[1] || [];

                            for(var j=0; j<newObjsList.length; j++) {
                                if(newObjsList[j].triplesContext != null) {
                                    triplesContext = triplesContext.concat(newObjsList[j].triplesContext);
                                    pairs.push([newVerb, newObjsList[j].chainSubject]);
                                } else {
                                    pairs.push([newVerb, newObjsList[j]])
                                }
                            }
                        }

                        token.pairs = pairs;
                        token.triplesContext = triplesContext;

                        return token;
                    })(result1[0], result1[2], result1[3])
                        : null;
                    if (result2 !== null) {
                        var result0 = result2;
                    } else {
                        var result0 = null;
                        pos = savedPos0;
                    }
                    reportMatchFailures = savedReportMatchFailures;
                    if (reportMatchFailures && result0 === null) {
                        matchFailed("[73] PropertyListNotEmptyPath");
                    }

                    cache[cacheKey] = {
                        nextPos: pos,
                        result:  result0
                    };
                    return result0;
                }

                function parse_PropertyListPath() {
                    var cacheKey = 'PropertyListPath@' + pos;
                    var cachedResult = cache[cacheKey];
                    if (cachedResult) {
                        pos = cachedResult.nextPos;
```

```
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result1 = parse_PropertyListNotEmpty();
        var result0 = result1 !== null ? result1 : '';
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[74] PropertyListPath");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_VerbPath() {
        var cacheKey = 'VerbPath@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var result1 = parse_PathAlternative();
        var result2 = result1 !== null
            ? (function(p) {
            var path = {};
            path.token = 'path';
            path.kind = 'element';
            path.value = p;

            return p;
        })(result1)
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[75]");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PathAlternative() {
        var cacheKey = 'PathAlternative@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }
```

```
            var savedReportMatchFailures = reportMatchFailures;
            reportMatchFailures = false;
            var savedPos0 = pos;
            var savedPos1 = pos;
            var result3 = parse_PathSequence();
            if (result3 !== null) {
                var result4 = [];
                var savedPos2 = pos;
                if (input.substr(pos, 1) === "|") {
                    var result6 = "|";
                    pos += 1;
                } else {
                    var result6 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"|\"");
                    }
                }
                if (result6 !== null) {
                    var result7 = parse_PathSequence();
                    if (result7 !== null) {
                        var result5 = [result6, result7];
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
                while (result5 !== null) {
                    result4.push(result5);
                    var savedPos2 = pos;
                    if (input.substr(pos, 1) === "|") {
                        var result6 = "|";
                        pos += 1;
                    } else {
                        var result6 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"|\"");
                        }
                    }
                    if (result6 !== null) {
                        var result7 = parse_PathSequence();
                        if (result7 !== null) {
                            var result5 = [result6, result7];
                        } else {
                            var result5 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                }
                if (result4 !== null) {
                    var result1 = [result3, result4];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
            var result2 = result1 !== null
                ? (function(first, rest) {
                if(rest == null || rest.length === 0) {
```

```
                return first;
            } else {
                var acum = [];
                for(var i=0; i<rest.length; i++)
                    acum.push(rest[1]);

                var path = {};
                path.token = 'path';
                path.kind = 'alternative';
                path.value = acum;

                return path;
            }
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[78] PathAlternative");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PathSequence() {
        var cacheKey = 'PathSequence@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_PathEltOrInverse();
        if (result3 !== null) {
            var result4 = [];
            var savedPos2 = pos;
            if (input.substr(pos, 1) === "/") {
                var result6 = "/";
                pos += 1;
            } else {
                var result6 = null;
                if (reportMatchFailures) {
                    matchFailed("\"/\"");
                }
            }
            if (result6 !== null) {
                var result7 = parse_PathEltOrInverse();
                if (result7 !== null) {
                    var result5 = [result6, result7];
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
```

```
            var result5 = null;
            pos = savedPos2;
        }
        while (result5 !== null) {
            result4.push(result5);
            var savedPos2 = pos;
            if (input.substr(pos, 1) === "/") {
                var result6 = "/";
                pos += 1;
            } else {
                var result6 = null;
                if (reportMatchFailures) {
                    matchFailed("\"/\"");
                }
            }
            if (result6 !== null) {
                var result7 = parse_PathEltOrInverse();
                if (result7 !== null) {
                    var result5 = [result6, result7];
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
        }
        if (result4 !== null) {
            var result1 = [result3, result4];
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(first, rest) {
        if(rest == null || rest.length === 0) {
            return first;
        } else {
            var acum = [first];

            for(var i=0; i<rest.length; i++)
                acum.push(rest[i][1]);

            var path = {};
            path.token = 'path';
            path.kind = 'sequence';

            path.value = acum;

            return path;
        }
    })(result1[0], result1[1])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[79] PathSequence");
```

```
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PathElt() {
        var cacheKey = 'PathElt@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_PathPrimary();
        if (result3 !== null) {
            var result5 = parse_PathMod();
            var result4 = result5 !== null ? result5 : '';
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(p, mod) {
            if(p.token && p.token != 'path' && mod == '') {
                return p;
            } else if(p.token && p.token != path && mod != '') {
                var path = {};
                path.token = 'path';
                path.kind = 'element';
                path.value = p;
                path.modifier = mod;
                return path;
            } else {
                p.modifier = mod;
                return p;
            }
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[88] PathElt");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
```

```
        return result0;
    }

    function parse_PathEltOrInverse() {
        var cacheKey = 'PathEltOrInverse@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result6 = parse_PathElt();
        if (result6 !== null) {
            var result0 = result6;
        } else {
            var savedPos0 = pos;
            var savedPos1 = pos;
            if (input.substr(pos, 1) === "^") {
                var result4 = "^";
                pos += 1;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("\"^\"");
                }
            }
            if (result4 !== null) {
                var result5 = parse_PathElt();
                if (result5 !== null) {
                    var result2 = [result4, result5];
                } else {
                    var result2 = null;
                    pos = savedPos1;
                }
            } else {
                var result2 = null;
                pos = savedPos1;
            }
            var result3 = result2 !== null
                ? (function(elt) {
                var path = {};
                path.token = 'path';
                path.kind = 'inversePath';
                path.value = elt;

                return path;
            })(result2[1])
                : null;
            if (result3 !== null) {
                var result1 = result3;
            } else {
                var result1 = null;
                pos = savedPos0;
            }
            if (result1 !== null) {
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[81] PathEltOrInverse");
        }
```

```
        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PathMod() {
        var cacheKey = 'PathMod@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        if (input.substr(pos, 1) === "*") {
            var result21 = "*";
            pos += 1;
        } else {
            var result21 = null;
            if (reportMatchFailures) {
                matchFailed("\"*\"");
            }
        }
        if (result21 !== null) {
            var result0 = result21;
        } else {
            if (input.substr(pos, 1) === "?") {
                var result20 = "?";
                pos += 1;
            } else {
                var result20 = null;
                if (reportMatchFailures) {
                    matchFailed("\"?\"");
                }
            }
            if (result20 !== null) {
                var result0 = result20;
            } else {
                if (input.substr(pos, 1) === "+") {
                    var result19 = "+";
                    pos += 1;
                } else {
                    var result19 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"+\"");
                    }
                }
                if (result19 !== null) {
                    var result0 = result19;
                } else {
                    var savedPos0 = pos;
                    if (input.substr(pos, 1) === "{") {
                        var result2 = "{";
                        pos += 1;
                    } else {
                        var result2 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"{\"");
                        }
                    }
                    if (result2 !== null) {
                        var savedPos2 = pos;
                        var result9 = parse_INTEGER();
                        if (result9 !== null) {
```

```
var savedPos3 = pos;
if (input.substr(pos, 1) === ",") {
    var result13 = ",";
    pos += 1;
} else {
    var result13 = null;
    if (reportMatchFailures) {
        matchFailed("\",\"");
    }
}
if (result13 !== null) {
    if (input.substr(pos, 1) === "}") {
        var result18 = "}";
        pos += 1;
    } else {
        var result18 = null;
        if (reportMatchFailures) {
            matchFailed("\"}\"");
        }
    }
    if (result18 !== null) {
        var result14 = result18;
    } else {
        var savedPos4 = pos;
        var result16 = parse_INTEGER();
        if (result16 !== null) {
            if (input.substr(pos, 1) === "}") {
                var result17 = "}";
                pos += 1;
            } else {
                var result17 = null;
                if (reportMatchFailures) {
                    matchFailed("\"}\"");
                }
            }
            if (result17 !== null) {
                var result15 = [result16, result17];
            } else {
                var result15 = null;
                pos = savedPos4;
            }
        } else {
            var result15 = null;
            pos = savedPos4;
        }
        if (result15 !== null) {
            var result14 = result15;
        } else {
            var result14 = null;;
        };
    }
    if (result14 !== null) {
        var result12 = [result13, result14];
    } else {
        var result12 = null;
        pos = savedPos3;
    }
} else {
    var result12 = null;
    pos = savedPos3;
}
if (result12 !== null) {
    var result10 = result12;
} else {
    if (input.substr(pos, 1) === "}") {
        var result11 = "}";
        pos += 1;
```

```
            } else {
                var result11 = null;
                if (reportMatchFailures) {
                    matchFailed("\"}\"");
                }
            }
            if (result11 !== null) {
                var result10 = result11;
            } else {
                var result10 = null;;
            };
        }
        if (result10 !== null) {
            var result8 = [result9, result10];
        } else {
            var result8 = null;
            pos = savedPos2;
        }
    } else {
        var result8 = null;
        pos = savedPos2;
    }
    if (result8 !== null) {
        var result3 = result8;
    } else {
        var savedPos1 = pos;
        if (input.substr(pos, 1) === ",") {
            var result5 = ",";
            pos += 1;
        } else {
            var result5 = null;
            if (reportMatchFailures) {
                matchFailed("\",\"");
            }
        }
        if (result5 !== null) {
            var result6 = parse_INTEGER();
            if (result6 !== null) {
                if (input.substr(pos, 1) === "}") {
                    var result7 = "}";
                    pos += 1;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"}\"");
                    }
                }
                if (result7 !== null) {
                    var result4 = [result5, result6, result7];
                } else {
                    var result4 = null;
                    pos = savedPos1;
                }
            } else {
                var result4 = null;
                pos = savedPos1;
            }
        } else {
            var result4 = null;
            pos = savedPos1;
        }
        if (result4 !== null) {
            var result3 = result4;
        } else {
            var result3 = null;;
        };
    }
```

```
                                if (result3 !== null) {
                                    var result1 = [result2, result3];
                                } else {
                                    var result1 = null;
                                    pos = savedPos0;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos0;
                            }
                            if (result1 !== null) {
                                var result0 = result1;
                            } else {
                                var result0 = null;;
                            };
                        };
                    };
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[82] PathMod");
                }

                cache[cacheKey] = {
                    nextPos: pos,
                    result:  result0
                };
                return result0;
            }

            function parse_PathPrimary() {
                var cacheKey = 'PathPrimary@' + pos;
                var cachedResult = cache[cacheKey];
                if (cachedResult) {
                    pos = cachedResult.nextPos;
                    return cachedResult.result;
                }

                var savedReportMatchFailures = reportMatchFailures;
                reportMatchFailures = false;
                var result13 = parse_IRIref();
                if (result13 !== null) {
                    var result0 = result13;
                } else {
                    var savedPos3 = pos;
                    if (input.substr(pos, 1) === "a") {
                        var result11 = "a";
                        pos += 1;
                    } else {
                        var result11 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"a\"");
                        }
                    }
                    var result12 = result11 !== null
                        ? (function() {
                        return{token: 'uri', prefix:null, suffix:null, value:"http://
www.w3.org/1999/02/22-rdf-syntax-ns#type"}
                    })()
                        : null;
                    if (result12 !== null) {
                        var result10 = result12;
                    } else {
                        var result10 = null;
                        pos = savedPos3;
                    }
                    if (result10 !== null) {
```

```
            var result0 = result10;
        } else {
            var savedPos2 = pos;
            if (input.substr(pos, 1) === "!") {
                var result8 = "!";
                pos += 1;
            } else {
                var result8 = null;
                if (reportMatchFailures) {
                    matchFailed("\"!\"");
                }
            }
            if (result8 !== null) {
                var result9 = parse_PathNegatedPropertySet();
                if (result9 !== null) {
                    var result7 = [result8, result9];
                } else {
                    var result7 = null;
                    pos = savedPos2;
                }
            } else {
                var result7 = null;
                pos = savedPos2;
            }
            if (result7 !== null) {
                var result0 = result7;
            } else {
                var savedPos0 = pos;
                var savedPos1 = pos;
                if (input.substr(pos, 1) === "(") {
                    var result4 = "(";
                    pos += 1;
                } else {
                    var result4 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
                }
                if (result4 !== null) {
                    var result5 = parse_PathAlternative();
                    if (result5 !== null) {
                        if (input.substr(pos, 1) === ")") {
                            var result6 = ")";
                            pos += 1;
                        } else {
                            var result6 = null;
                            if (reportMatchFailures) {
                                matchFailed("\")\"");
                            }
                        }
                        if (result6 !== null) {
                            var result2 = [result4, result5, result6];
                        } else {
                            var result2 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result2 = null;
                    pos = savedPos1;
                }
                var result3 = result2 !== null
                    ? (function(p) {
                    return p;
```

```
                    })(result2[1])
                        : null;
                    if (result3 !== null) {
                        var result1 = result3;
                    } else {
                        var result1 = null;
                        pos = savedPos0;
                    }
                    if (result1 !== null) {
                        var result0 = result1;
                    } else {
                        var result0 = null;;
                    };
                };
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[83] PathPrimary");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PathNegatedPropertySet() {
        var cacheKey = 'PathNegatedPropertySet@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }


        var result11 = parse_PathOneInPropertySet();
        if (result11 !== null) {
            var result0 = result11;
        } else {
            var savedPos0 = pos;
            if (input.substr(pos, 1) === "(") {
                var result2 = "(";
                pos += 1;
            } else {
                var result2 = null;
                if (reportMatchFailures) {
                    matchFailed("\"(\"");
                }
            }
            if (result2 !== null) {
                var savedPos1 = pos;
                var result6 = parse_PathOneInPropertySet();
                if (result6 !== null) {
                    var result7 = [];
                    var savedPos2 = pos;
                    if (input.substr(pos, 1) === "|") {
                        var result9 = "|";
                        pos += 1;
                    } else {
                        var result9 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"|\"");
                        }
                    }
                    if (result9 !== null) {
```

```
            var result10 = parse_PathOneInPropertySet();
            if (result10 !== null) {
                var result8 = [result9, result10];
            } else {
                var result8 = null;
                pos = savedPos2;
            }
        } else {
            var result8 = null;
            pos = savedPos2;
        }
        while (result8 !== null) {
            result7.push(result8);
            var savedPos2 = pos;
            if (input.substr(pos, 1) === "|") {
                var result9 = "|";
                pos += 1;
            } else {
                var result9 = null;
                if (reportMatchFailures) {
                    matchFailed("\"|\"");
                }
            }
            if (result9 !== null) {
                var result10 = parse_PathOneInPropertySet();
                if (result10 !== null) {
                    var result8 = [result9, result10];
                } else {
                    var result8 = null;
                    pos = savedPos2;
                }
            } else {
                var result8 = null;
                pos = savedPos2;
            }
        }
        if (result7 !== null) {
            var result5 = [result6, result7];
        } else {
            var result5 = null;
            pos = savedPos1;
        }
    } else {
        var result5 = null;
        pos = savedPos1;
    }
    var result3 = result5 !== null ? result5 : '';
    if (result3 !== null) {
        if (input.substr(pos, 1) === ")") {
            var result4 = ")";
            pos += 1;
        } else {
            var result4 = null;
            if (reportMatchFailures) {
                matchFailed("\")\"");
            }
        }
        if (result4 !== null) {
            var result1 = [result2, result3, result4];
        } else {
            var result1 = null;
            pos = savedPos0;
        }
    } else {
        var result1 = null;
        pos = savedPos0;
    }
```

```
            } else {
                var result1 = null;
                pos = savedPos0;
            }
            if (result1 !== null) {
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        }



        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PathOneInPropertySet() {
        var cacheKey = 'PathOneInPropertySet@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result7 = parse_IRIref();
        if (result7 !== null) {
            var result0 = result7;
        } else {
            if (input.substr(pos, 1) === "a") {
                var result6 = "a";
                pos += 1;
            } else {
                var result6 = null;
                if (reportMatchFailures) {
                    matchFailed("\"a\"");
                }
            }
            if (result6 !== null) {
                var result0 = result6;
            } else {
                var savedPos0 = pos;
                if (input.substr(pos, 1) === "^") {
                    var result2 = "^";
                    pos += 1;
                } else {
                    var result2 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"^\"");
                    }
                }
                if (result2 !== null) {
                    var result5 = parse_IRIref();
                    if (result5 !== null) {
                        var result3 = result5;
                    } else {
                        if (input.substr(pos, 1) === "a") {
                            var result4 = "a";
                            pos += 1;
                        } else {
                            var result4 = null;
                            if (reportMatchFailures) {
```

```
                                matchFailed("\"a\"");
                            }
                        }
                        if (result4 !== null) {
                            var result3 = result4;
                        } else {
                            var result3 = null;;
                        };
                    }
                    if (result3 !== null) {
                        var result1 = [result2, result3];
                    } else {
                        var result1 = null;
                        pos = savedPos0;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos0;
                }
                if (result1 !== null) {
                    var result0 = result1;
                } else {
                    var result0 = null;;
                };
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[85] PathOneInPropertySet");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_TriplesNode() {
        var cacheKey = 'TriplesNode@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var result3 = parse_Collection();
        var result4 = result3 !== null
            ? (function(c) {
            var triplesContext = [];
            var chainSubject = [];

            var triple = null;

            // catch NIL
            /*
             if(c.length == 1 && c[0].token && c[0].token === 'nil') {
             GlobalBlankNodeCounter++;
             return  {token: "triplesnodecollection",
             triplesContext:[{subject: {token:'blank', value:("_:"+GlobalBlankNodeCounter)},
             predicate:{token:'uri', prefix:null, suffix:null, value:'http://
www.w3.org/1999/02/22-rdf-syntax-ns#rest'},
             object:  {token:'blank', value:("_:"+(GlobalBlankNodeCounter+1))}}],
             chainSubject:{token:'blank', value:("_:"+GlobalBlankNodeCounter)}};
```

```
                        }
                        */

                       // other cases
                       for(var i=0; i<c.length; i++) {
                           GlobalBlankNodeCounter++;
                           //_:b0  rdf:first  1 ;
                           //rdf:rest   _:b1 .
                           var nextObject = null;
                           if(c[i].chainSubject == null && c[i].triplesContext == null) {
                               nextObject = c[i];
                           } else {
                               nextObject = c[i].chainSubject;
                               triplesContext = triplesContext.concat(nextSubject.triplesContext);
                           }
                           var currentSubject = null;
                           triple = {subject: {token:'blank', value:("_:"+GlobalBlankNodeCounter)},
                               predicate:{token:'uri', prefix:null, suffix:null, value:'http://
www.w3.org/1999/02/22-rdf-syntax-ns#first'},
                               object:nextObject };

                           if(i==0) {
                               chainSubject.push(triple.subject);
                           }

                           triplesContext.push(triple);

                           if(i===(c.length-1)) {
                               triple = {subject: {token:'blank', value:("_:"+GlobalBlankNodeCounter)},
                                   predicate:{token:'uri', prefix:null, suffix:null, value:'http://
www.w3.org/1999/02/22-rdf-syntax-ns#rest'},
                                   object:   {token:'uri', prefix:null, suffix:null, value:'http://
www.w3.org/1999/02/22-rdf-syntax-ns#nil'}};
                           } else {
                               triple = {subject: {token:'blank', value:("_:"+GlobalBlankNodeCounter)},
                                   predicate:{token:'uri', prefix:null, suffix:null, value:'http://
www.w3.org/1999/02/22-rdf-syntax-ns#rest'},
                                   object:  {token:'blank', value:("_:"+(GlobalBlankNodeCounter+1))} };
                           }

                           triplesContext.push(triple);
                       }

                           return {token:"triplesnodecollection", triplesContext:triplesContext,
chainSubject:chainSubject};
                   })(result3)
                       : null;
                   if (result4 !== null) {
                       var result2 = result4;
                   } else {
                       var result2 = null;
                       pos = savedPos0;
                   }
                   if (result2 !== null) {
                       var result0 = result2;
                   } else {
                       var result1 = parse_BlankNodePropertyList();
                       if (result1 !== null) {
                           var result0 = result1;
                       } else {
                           var result0 = null;;
                       };
                   }
                   reportMatchFailures = savedReportMatchFailures;
                   if (reportMatchFailures && result0 === null) {
                       matchFailed("[87] TriplesNode");
```

```
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_BlankNodePropertyList() {
        var cacheKey = 'BlankNodePropertyList@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result13 = parse_WS();
        while (result13 !== null) {
            result3.push(result13);
            var result13 = parse_WS();
        }
        if (result3 !== null) {
            if (input.substr(pos, 1) === "[") {
                var result4 = "[";
                pos += 1;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("\"[\"");
                }
            }
            if (result4 !== null) {
                var result5 = [];
                var result12 = parse_WS();
                while (result12 !== null) {
                    result5.push(result12);
                    var result12 = parse_WS();
                }
                if (result5 !== null) {
                    var result6 = parse_PropertyListNotEmpty();
                    if (result6 !== null) {
                        var result7 = [];
                        var result11 = parse_WS();
                        while (result11 !== null) {
                            result7.push(result11);
                            var result11 = parse_WS();
                        }
                        if (result7 !== null) {
                            if (input.substr(pos, 1) === "]") {
                                var result8 = "]";
                                pos += 1;
                            } else {
                                var result8 = null;
                                if (reportMatchFailures) {
                                    matchFailed("\"]\"");
                                }
                            }
                            if (result8 !== null) {
                                var result9 = [];
                                var result10 = parse_WS();
                                while (result10 !== null) {
```

```
                                        result9.push(result10);
                                        var result10 = parse_WS();
                                    }
                                    if (result9 !== null) {
                                        var result1 = [result3, result4, result5, result6,
result7, result8, result9];
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function(pl) {

                    GlobalBlankNodeCounter++;
                    var subject = {token:'blank', value:'_:'+GlobalBlankNodeCounter};
                    var newTriples =  [];

                    for(var i=0; i< pl.pairs.length; i++) {
                        var pair = pl.pairs[i];
                        var triple = {}
                        triple.subject = subject;
                        triple.predicate = pair[0];
                        if(pair[1].length != null)
                            pair[1] = pair[1][0]
                        triple.object = pair[1];
                        newTriples.push(triple);
                    }

                    return {token: 'triplesnode',
                        kind: 'blanknodepropertylist',
                        triplesContext: pl.triplesContext.concat(newTriples),
                        chainSubject: subject};
                })(result1[3])
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[88] BlankNodePropertyList");
```

```
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Collection() {
        var cacheKey = 'Collection@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = [];
        var result14 = parse_WS();
        while (result14 !== null) {
            result3.push(result14);
            var result14 = parse_WS();
        }
        if (result3 !== null) {
            if (input.substr(pos, 1) === "(") {
                var result4 = "(";
                pos += 1;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("\"(\"");
                }
            }
            if (result4 !== null) {
                var result5 = [];
                var result13 = parse_WS();
                while (result13 !== null) {
                    result5.push(result13);
                    var result13 = parse_WS();
                }
                if (result5 !== null) {
                    var result12 = parse_GraphNode();
                    if (result12 !== null) {
                        var result6 = [];
                        while (result12 !== null) {
                            result6.push(result12);
                            var result12 = parse_GraphNode();
                        }
                    } else {
                        var result6 = null;
                    }
                    if (result6 !== null) {
                        var result7 = [];
                        var result11 = parse_WS();
                        while (result11 !== null) {
                            result7.push(result11);
                            var result11 = parse_WS();
                        }
                        if (result7 !== null) {
                            if (input.substr(pos, 1) === ")") {
                                var result8 = ")";
                                pos += 1;
                            } else {
```

```
                                        var result8 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("\")\"");
                                        }
                                    }
                                    if (result8 !== null) {
                                        var result9 = [];
                                        var result10 = parse_WS();
                                        while (result10 !== null) {
                                            result9.push(result10);
                                            var result10 = parse_WS();
                                        }
                                        if (result9 !== null) {
                                            var result1 = [result3, result4, result5, result6,
result7, result8, result9];
                                        } else {
                                            var result1 = null;
                                            pos = savedPos1;
                                        }
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                    var result2 = result1 !== null
                        ? (function(gn) {
                        return gn;
                    })(result1[3])
                        : null;
                    if (result2 !== null) {
                        var result0 = result2;
                    } else {
                        var result0 = null;
                        pos = savedPos0;
                    }
                    reportMatchFailures = savedReportMatchFailures;
                    if (reportMatchFailures && result0 === null) {
                        matchFailed("[89] Collection");
                    }

                    cache[cacheKey] = {
                        nextPos: pos,
                        result:  result0
                    };
                    return result0;
                }

                function parse_GraphNode() {
```

```
var cacheKey = 'GraphNode@' + pos;
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos2 = pos;
var result10 = [];
var result14 = parse_WS();
while (result14 !== null) {
    result10.push(result14);
    var result14 = parse_WS();
}
if (result10 !== null) {
    var result11 = parse_VarOrTerm();
    if (result11 !== null) {
        var result12 = [];
        var result13 = parse_WS();
        while (result13 !== null) {
            result12.push(result13);
            var result13 = parse_WS();
        }
        if (result12 !== null) {
            var result9 = [result10, result11, result12];
        } else {
            var result9 = null;
            pos = savedPos2;
        }
    } else {
        var result9 = null;
        pos = savedPos2;
    }
} else {
    var result9 = null;
    pos = savedPos2;
}
if (result9 !== null) {
    var result1 = result9;
} else {
    var savedPos1 = pos;
    var result4 = [];
    var result8 = parse_WS();
    while (result8 !== null) {
        result4.push(result8);
        var result8 = parse_WS();
    }
    if (result4 !== null) {
        var result5 = parse_TriplesNode();
        if (result5 !== null) {
            var result6 = [];
            var result7 = parse_WS();
            while (result7 !== null) {
                result6.push(result7);
                var result7 = parse_WS();
            }
            if (result6 !== null) {
                var result3 = [result4, result5, result6];
            } else {
                var result3 = null;
                pos = savedPos1;
            }
        } else {
            var result3 = null;
```

```
                    pos = savedPos1;
                }
            } else {
                var result3 = null;
                pos = savedPos1;
            }
            if (result3 !== null) {
                var result1 = result3;
            } else {
                var result1 = null;;
            };
        }
        var result2 = result1 !== null
            ? (function(gn) {
            return gn[1];
        })(result1)
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[90] GraphNode");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_VarOrTerm() {
        var cacheKey = 'VarOrTerm@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result2 = parse_Var();
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result1 = parse_GraphTerm();
            if (result1 !== null) {
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[91] VarOrTerm");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }
```

```
function parse_VarOrIRIref() {
    var cacheKey = 'VarOrIRIref@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var result2 = parse_Var();
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result1 = parse_IRIref();
        if (result1 !== null) {
            var result0 = result1;
        } else {
            var result0 = null;;
        };
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[92] VarOrIRIref");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_Var() {
    var cacheKey = 'Var@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos0 = pos;
    var result4 = parse_VAR1();
    if (result4 !== null) {
        var result1 = result4;
    } else {
        var result3 = parse_VAR2();
        if (result3 !== null) {
            var result1 = result3;
        } else {
            var result1 = null;;
        };
    }
    var result2 = result1 !== null
        ? (function(v) {
        var term = {};
        term.token = 'var';
        term.value = v;
        return term;
    })(result1)
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
```

```
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[93] Var");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_GraphTerm() {
        var cacheKey = 'GraphTerm@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result6 = parse_IRIref();
        if (result6 !== null) {
            var result0 = result6;
        } else {
            var result5 = parse_RDFLiteral();
            if (result5 !== null) {
                var result0 = result5;
            } else {
                var result4 = parse_NumericLiteral();
                if (result4 !== null) {
                    var result0 = result4;
                } else {
                    var result3 = parse_BooleanLiteral();
                    if (result3 !== null) {
                        var result0 = result3;
                    } else {
                        var result2 = parse_BlankNode();
                        if (result2 !== null) {
                            var result0 = result2;
                        } else {
                            var result1 = parse_NIL();
                            if (result1 !== null) {
                                var result0 = result1;
                            } else {
                                var result0 = null;;
                            };
                        };
                    };
                };
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[94] GraphTerm");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }
```

```
        function parse_ConditionalOrExpression() {
            var cacheKey = 'ConditionalOrExpression@' + pos;
            var cachedResult = cache[cacheKey];
            if (cachedResult) {
                pos = cachedResult.nextPos;
                return cachedResult.result;
            }

            var savedReportMatchFailures = reportMatchFailures;
            reportMatchFailures = false;
            var savedPos0 = pos;
            var savedPos1 = pos;
            var result3 = parse_ConditionalAndExpression();
            if (result3 !== null) {
                var result4 = [];
                var savedPos2 = pos;
                var result6 = [];
                var result11 = parse_WS();
                while (result11 !== null) {
                    result6.push(result11);
                    var result11 = parse_WS();
                }
                if (result6 !== null) {
                    if (input.substr(pos, 2) === "||") {
                        var result7 = "||";
                        pos += 2;
                    } else {
                        var result7 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"||\"");
                        }
                    }
                    if (result7 !== null) {
                        var result8 = [];
                        var result10 = parse_WS();
                        while (result10 !== null) {
                            result8.push(result10);
                            var result10 = parse_WS();
                        }
                        if (result8 !== null) {
                            var result9 = parse_ConditionalAndExpression();
                            if (result9 !== null) {
                                var result5 = [result6, result7, result8, result9];
                            } else {
                                var result5 = null;
                                pos = savedPos2;
                            }
                        } else {
                            var result5 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
                while (result5 !== null) {
                    result4.push(result5);
                    var savedPos2 = pos;
                    var result6 = [];
                    var result11 = parse_WS();
                    while (result11 !== null) {
                        result6.push(result11);
```

```
                var result11 = parse_WS();
            }
            if (result6 !== null) {
                if (input.substr(pos, 2) === "||") {
                    var result7 = "||";
                    pos += 2;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"||\"");
                    }
                }
                if (result7 !== null) {
                    var result8 = [];
                    var result10 = parse_WS();
                    while (result10 !== null) {
                        result8.push(result10);
                        var result10 = parse_WS();
                    }
                    if (result8 !== null) {
                        var result9 = parse_ConditionalAndExpression();
                        if (result9 !== null) {
                            var result5 = [result6, result7, result8, result9];
                        } else {
                            var result5 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
        }
        if (result4 !== null) {
            var result1 = [result3, result4];
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(v, vs) {
        if(vs.length === 0) {
            return v;
        }

        var exp = {};
        exp.token = "expression";
        exp.expressionType = "conditionalor";
        var ops = [v];

        for(var i=0; i<vs.length; i++) {
            ops.push(vs[i][3]);
        }

        exp.operands = ops;
```

```
            return exp;
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[96] ConditionalOrExpression");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ConditionalAndExpression() {
        var cacheKey = 'ConditionalAndExpression@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_RelationalExpression();
        if (result3 !== null) {
            var result4 = [];
            var savedPos2 = pos;
            var result6 = [];
            var result11 = parse_WS();
            while (result11 !== null) {
                result6.push(result11);
                var result11 = parse_WS();
            }
            if (result6 !== null) {
                if (input.substr(pos, 2) === "&&") {
                    var result7 = "&&";
                    pos += 2;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"&&\"");
                    }
                }
                if (result7 !== null) {
                    var result8 = [];
                    var result10 = parse_WS();
                    while (result10 !== null) {
                        result8.push(result10);
                        var result10 = parse_WS();
                    }
                    if (result8 !== null) {
                        var result9 = parse_RelationalExpression();
                        if (result9 !== null) {
                            var result5 = [result6, result7, result8, result9];
                        } else {
                            var result5 = null;
                            pos = savedPos2;
```

```
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
        } else {
            var result5 = null;
            pos = savedPos2;
        }
        while (result5 !== null) {
            result4.push(result5);
            var savedPos2 = pos;
            var result6 = [];
            var result11 = parse_WS();
            while (result11 !== null) {
                result6.push(result11);
                var result11 = parse_WS();
            }
            if (result6 !== null) {
                if (input.substr(pos, 2) === "&&") {
                    var result7 = "&&";
                    pos += 2;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"&&\"");
                    }
                }
                if (result7 !== null) {
                    var result8 = [];
                    var result10 = parse_WS();
                    while (result10 !== null) {
                        result8.push(result10);
                        var result10 = parse_WS();
                    }
                    if (result8 !== null) {
                        var result9 = parse_RelationalExpression();
                        if (result9 !== null) {
                            var result5 = [result6, result7, result8, result9];
                        } else {
                            var result5 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result5 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result5 = null;
                    pos = savedPos2;
                }
            } else {
                var result5 = null;
                pos = savedPos2;
            }
        }
        if (result4 !== null) {
            var result1 = [result3, result4];
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
```

```
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(v, vs) {
            if(vs.length === 0) {
                return v;
            }
            var exp = {};
            exp.token = "expression";
            exp.expressionType = "conditionaland";
            var ops = [v];

            for(var i=0; i<vs.length; i++) {
                ops.push(vs[i][3]);
            }

            exp.operands = ops;

            return exp;
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[97] ConditionalAndExpression");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_RelationalExpression() {
        var cacheKey = 'RelationalExpression@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_AdditiveExpression();
        if (result3 !== null) {
            var result4 = [];
            var savedPos7 = pos;
            var result42 = [];
            var result47 = parse_WS();
            while (result47 !== null) {
                result42.push(result47);
                var result47 = parse_WS();
            }
            if (result42 !== null) {
                if (input.substr(pos, 1) === "=") {
                    var result43 = "=";
                    pos += 1;
                } else {
```

```
                var result43 = null;
                if (reportMatchFailures) {
                    matchFailed("\"=\"");
                }
            }
            if (result43 !== null) {
                var result44 = [];
                var result46 = parse_WS();
                while (result46 !== null) {
                    result44.push(result46);
                    var result46 = parse_WS();
                }
                if (result44 !== null) {
                    var result45 = parse_AdditiveExpression();
                    if (result45 !== null) {
                        var result41 = [result42, result43, result44, result45];
                    } else {
                        var result41 = null;
                        pos = savedPos7;
                    }
                } else {
                    var result41 = null;
                    pos = savedPos7;
                }
            } else {
                var result41 = null;
                pos = savedPos7;
            }
        } else {
            var result41 = null;
            pos = savedPos7;
        }
        if (result41 !== null) {
            var result5 = result41;
        } else {
            var savedPos6 = pos;
            var result35 = [];
            var result40 = parse_WS();
            while (result40 !== null) {
                result35.push(result40);
                var result40 = parse_WS();
            }
            if (result35 !== null) {
                if (input.substr(pos, 2) === "!=") {
                    var result36 = "!=";
                    pos += 2;
                } else {
                    var result36 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"!=\"");
                    }
                }
                if (result36 !== null) {
                    var result37 = [];
                    var result39 = parse_WS();
                    while (result39 !== null) {
                        result37.push(result39);
                        var result39 = parse_WS();
                    }
                    if (result37 !== null) {
                        var result38 = parse_AdditiveExpression();
                        if (result38 !== null) {
                            var result34 = [result35, result36, result37, result38];
                        } else {
                            var result34 = null;
                            pos = savedPos6;
                        }
```

```
            } else {
                var result34 = null;
                pos = savedPos6;
            }
        } else {
            var result34 = null;
            pos = savedPos6;
        }
    } else {
        var result34 = null;
        pos = savedPos6;
    }
    if (result34 !== null) {
        var result5 = result34;
    } else {
        var savedPos5 = pos;
        var result28 = [];
        var result33 = parse_WS();
        while (result33 !== null) {
            result28.push(result33);
            var result33 = parse_WS();
        }
        if (result28 !== null) {
            if (input.substr(pos, 1) === "<") {
                var result29 = "<";
                pos += 1;
            } else {
                var result29 = null;
                if (reportMatchFailures) {
                    matchFailed("\"<\"");
                }
            }
            if (result29 !== null) {
                var result30 = [];
                var result32 = parse_WS();
                while (result32 !== null) {
                    result30.push(result32);
                    var result32 = parse_WS();
                }
                if (result30 !== null) {
                    var result31 = parse_AdditiveExpression();
                    if (result31 !== null) {
                        var result27 = [result28, result29, result30, result31];
                    } else {
                        var result27 = null;
                        pos = savedPos5;
                    }
                } else {
                    var result27 = null;
                    pos = savedPos5;
                }
            } else {
                var result27 = null;
                pos = savedPos5;
            }
        } else {
            var result27 = null;
            pos = savedPos5;
        }
        if (result27 !== null) {
            var result5 = result27;
        } else {
            var savedPos4 = pos;
            var result21 = [];
            var result26 = parse_WS();
            while (result26 !== null) {
                result21.push(result26);
```

```
                                    var result26 = parse_WS();
                                }
                                if (result21 !== null) {
                                    if (input.substr(pos, 1) === ">") {
                                        var result22 = ">";
                                        pos += 1;
                                    } else {
                                        var result22 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("\">\"");
                                        }
                                    }
                                    if (result22 !== null) {
                                        var result23 = [];
                                        var result25 = parse_WS();
                                        while (result25 !== null) {
                                            result23.push(result25);
                                            var result25 = parse_WS();
                                        }
                                        if (result23 !== null) {
                                            var result24 = parse_AdditiveExpression();
                                            if (result24 !== null) {
                                                var result20 = [result21, result22, result23,
result24];
                                            } else {
                                                var result20 = null;
                                                pos = savedPos4;
                                            }
                                        } else {
                                            var result20 = null;
                                            pos = savedPos4;
                                        }
                                    } else {
                                        var result20 = null;
                                        pos = savedPos4;
                                    }
                                } else {
                                    var result20 = null;
                                    pos = savedPos4;
                                }
                                if (result20 !== null) {
                                    var result5 = result20;
                                } else {
                                    var savedPos3 = pos;
                                    var result14 = [];
                                    var result19 = parse_WS();
                                    while (result19 !== null) {
                                        result14.push(result19);
                                        var result19 = parse_WS();
                                    }
                                    if (result14 !== null) {
                                        if (input.substr(pos, 2) === "<=") {
                                            var result15 = "<=";
                                            pos += 2;
                                        } else {
                                            var result15 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\"<=\"");
                                            }
                                        }
                                        if (result15 !== null) {
                                            var result16 = [];
                                            var result18 = parse_WS();
                                            while (result18 !== null) {
                                                result16.push(result18);
                                                var result18 = parse_WS();
                                            }
```

```
                                        if (result16 !== null) {
                                            var result17 = parse_AdditiveExpression();
                                            if (result17 !== null) {
                                                var result13 = [result14, result15, result16,
result17];
                                            } else {
                                                var result13 = null;
                                                pos = savedPos3;
                                            }
                                        } else {
                                            var result13 = null;
                                            pos = savedPos3;
                                        }
                                    } else {
                                        var result13 = null;
                                        pos = savedPos3;
                                    }
                                } else {
                                    var result13 = null;
                                    pos = savedPos3;
                                }
                                if (result13 !== null) {
                                    var result5 = result13;
                                } else {
                                    var savedPos2 = pos;
                                    var result7 = [];
                                    var result12 = parse_WS();
                                    while (result12 !== null) {
                                        result7.push(result12);
                                        var result12 = parse_WS();
                                    }
                                    if (result7 !== null) {
                                        if (input.substr(pos, 2) === ">=") {
                                            var result8 = ">=";
                                            pos += 2;
                                        } else {
                                            var result8 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\">=\"");
                                            }
                                        }
                                        if (result8 !== null) {
                                            var result9 = [];
                                            var result11 = parse_WS();
                                            while (result11 !== null) {
                                                result9.push(result11);
                                                var result11 = parse_WS();
                                            }
                                            if (result9 !== null) {
                                                var result10 = parse_AdditiveExpression();
                                                if (result10 !== null) {
                                                    var result6 = [result7, result8, result9,
result10];
                                                } else {
                                                    var result6 = null;
                                                    pos = savedPos2;
                                                }
                                            } else {
                                                var result6 = null;
                                                pos = savedPos2;
                                            }
                                        } else {
                                            var result6 = null;
                                            pos = savedPos2;
                                        }
                                    } else {
                                        var result6 = null;
```

```
                            pos = savedPos2;
                        }
                        if (result6 !== null) {
                            var result5 = result6;
                        } else {
                            var result5 = null;;
                        };
                    };
                };
            };
        };
    }
    while (result5 !== null) {
        result4.push(result5);
        var savedPos7 = pos;
        var result42 = [];
        var result47 = parse_WS();
        while (result47 !== null) {
            result42.push(result47);
            var result47 = parse_WS();
        }
        if (result42 !== null) {
            if (input.substr(pos, 1) === "=") {
                var result43 = "=";
                pos += 1;
            } else {
                var result43 = null;
                if (reportMatchFailures) {
                    matchFailed("\"=\"");
                }
            }
            if (result43 !== null) {
                var result44 = [];
                var result46 = parse_WS();
                while (result46 !== null) {
                    result44.push(result46);
                    var result46 = parse_WS();
                }
                if (result44 !== null) {
                    var result45 = parse_AdditiveExpression();
                    if (result45 !== null) {
                        var result41 = [result42, result43, result44, result45];
                    } else {
                        var result41 = null;
                        pos = savedPos7;
                    }
                } else {
                    var result41 = null;
                    pos = savedPos7;
                }
            } else {
                var result41 = null;
                pos = savedPos7;
            }
        } else {
            var result41 = null;
            pos = savedPos7;
        }
        if (result41 !== null) {
            var result5 = result41;
        } else {
            var savedPos6 = pos;
            var result35 = [];
            var result40 = parse_WS();
            while (result40 !== null) {
                result35.push(result40);
                var result40 = parse_WS();
```

```
    }
    if (result35 !== null) {
        if (input.substr(pos, 2) === "!=") {
            var result36 = "!=";
            pos += 2;
        } else {
            var result36 = null;
            if (reportMatchFailures) {
                matchFailed("\"!=\"");
            }
        }
        if (result36 !== null) {
            var result37 = [];
            var result39 = parse_WS();
            while (result39 !== null) {
                result37.push(result39);
                var result39 = parse_WS();
            }
            if (result37 !== null) {
                var result38 = parse_AdditiveExpression();
                if (result38 !== null) {
                    var result34 = [result35, result36, result37, result38];
                } else {
                    var result34 = null;
                    pos = savedPos6;
                }
            } else {
                var result34 = null;
                pos = savedPos6;
            }
        } else {
            var result34 = null;
            pos = savedPos6;
        }
    } else {
        var result34 = null;
        pos = savedPos6;
    }
    if (result34 !== null) {
        var result5 = result34;
    } else {
        var savedPos5 = pos;
        var result28 = [];
        var result33 = parse_WS();
        while (result33 !== null) {
            result28.push(result33);
            var result33 = parse_WS();
        }
        if (result28 !== null) {
            if (input.substr(pos, 1) === "<") {
                var result29 = "<";
                pos += 1;
            } else {
                var result29 = null;
                if (reportMatchFailures) {
                    matchFailed("\"<\"");
                }
            }
            if (result29 !== null) {
                var result30 = [];
                var result32 = parse_WS();
                while (result32 !== null) {
                    result30.push(result32);
                    var result32 = parse_WS();
                }
                if (result30 !== null) {
                    var result31 = parse_AdditiveExpression();
```

```
                                        if (result31 !== null) {
                                            var result27 = [result28, result29, result30,
result31];
                                        } else {
                                            var result27 = null;
                                            pos = savedPos5;
                                        }
                                    } else {
                                        var result27 = null;
                                        pos = savedPos5;
                                    }
                                } else {
                                    var result27 = null;
                                    pos = savedPos5;
                                }
                                if (result27 !== null) {
                                    var result5 = result27;
                                } else {
                                    var savedPos4 = pos;
                                    var result21 = [];
                                    var result26 = parse_WS();
                                    while (result26 !== null) {
                                        result21.push(result26);
                                        var result26 = parse_WS();
                                    }
                                    if (result21 !== null) {
                                        if (input.substr(pos, 1) === ">") {
                                            var result22 = ">";
                                            pos += 1;
                                        } else {
                                            var result22 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\">\"");
                                            }
                                        }
                                        if (result22 !== null) {
                                            var result23 = [];
                                            var result25 = parse_WS();
                                            while (result25 !== null) {
                                                result23.push(result25);
                                                var result25 = parse_WS();
                                            }
                                            if (result23 !== null) {
                                                var result24 = parse_AdditiveExpression();
                                                if (result24 !== null) {
                                                    var result20 = [result21, result22, result23,
result24];
                                                } else {
                                                    var result20 = null;
                                                    pos = savedPos4;
                                                }
                                            } else {
                                                var result20 = null;
                                                pos = savedPos4;
                                            }
                                        } else {
                                            var result20 = null;
                                            pos = savedPos4;
                                        }
                                    } else {
                                        var result20 = null;
                                        pos = savedPos4;
                                    }
```

```
                                    if (result20 !== null) {
                                        var result5 = result20;
                                    } else {
                                        var savedPos3 = pos;
                                        var result14 = [];
                                        var result19 = parse_WS();
                                        while (result19 !== null) {
                                            result14.push(result19);
                                            var result19 = parse_WS();
                                        }
                                        if (result14 !== null) {
                                            if (input.substr(pos, 2) === "<=") {
                                                var result15 = "<=";
                                                pos += 2;
                                            } else {
                                                var result15 = null;
                                                if (reportMatchFailures) {
                                                    matchFailed("\"<=\"");
                                                }
                                            }
                                            if (result15 !== null) {
                                                var result16 = [];
                                                var result18 = parse_WS();
                                                while (result18 !== null) {
                                                    result16.push(result18);
                                                    var result18 = parse_WS();
                                                }
                                                if (result16 !== null) {
                                                    var result17 = parse_AdditiveExpression();
                                                    if (result17 !== null) {
                                                        var result13 = [result14, result15, result16,
result17];

                                                    } else {
                                                        var result13 = null;
                                                        pos = savedPos3;
                                                    }
                                                } else {
                                                    var result13 = null;
                                                    pos = savedPos3;
                                                }
                                            } else {
                                                var result13 = null;
                                                pos = savedPos3;
                                            }
                                        } else {
                                            var result13 = null;
                                            pos = savedPos3;
                                        }
                                        if (result13 !== null) {
                                            var result5 = result13;
                                        } else {
                                            var savedPos2 = pos;
                                            var result7 = [];
                                            var result12 = parse_WS();
                                            while (result12 !== null) {
                                                result7.push(result12);
                                                var result12 = parse_WS();
                                            }
                                            if (result7 !== null) {
                                                if (input.substr(pos, 2) === ">=") {
                                                    var result8 = ">=";
                                                    pos += 2;
                                                } else {
                                                    var result8 = null;
                                                    if (reportMatchFailures) {
                                                        matchFailed("\">=\"");
                                                    }
```

```
                                                }
                                                if (result8 !== null) {
                                                    var result9 = [];
                                                    var result11 = parse_WS();
                                                    while (result11 !== null) {
                                                        result9.push(result11);
                                                        var result11 = parse_WS();
                                                    }
                                                    if (result9 !== null) {
                                                        var result10 = parse_AdditiveExpression();
                                                        if (result10 !== null) {
                                                            var result6 = [result7, result8, result9,
result10];

                                                        } else {
                                                            var result6 = null;
                                                            pos = savedPos2;
                                                        }
                                                    } else {
                                                        var result6 = null;
                                                        pos = savedPos2;
                                                    }
                                                } else {
                                                    var result6 = null;
                                                    pos = savedPos2;
                                                }
                                            } else {
                                                var result6 = null;
                                                pos = savedPos2;
                                            }
                                            if (result6 !== null) {
                                                var result5 = result6;
                                            } else {
                                                var result5 = null;;
                                            };
                                        };
                                    };
                                };
                            }
                        }
                        if (result4 !== null) {
                            var result1 = [result3, result4];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                    var result2 = result1 !== null
                        ? (function(op1, op2) {
                        if(op2.length === 0) {
                            return op1;
                        } else {
                            var exp = {};
                            exp.expressionType = "relationalexpression"
                            exp.operator = op2[0][1];
                            exp.op1 = op1;
                            exp.op2 = op2[0][3];
                            exp.token = "expression";

                            return exp;
                        }
                    })(result1[0], result1[1])
                        : null;
                    if (result2 !== null) {
```

```
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[99] RelationalExpression");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_AdditiveExpression() {
        var cacheKey = 'AdditiveExpression@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_MultiplicativeExpression();
        if (result3 !== null) {
            var result4 = [];
            var savedPos6 = pos;
            var result34 = [];
            var result39 = parse_WS();
            while (result39 !== null) {
                result34.push(result39);
                var result39 = parse_WS();
            }
            if (result34 !== null) {
                if (input.substr(pos, 1) === "+") {
                    var result35 = "+";
                    pos += 1;
                } else {
                    var result35 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"+\"");
                    }
                }
                if (result35 !== null) {
                    var result36 = [];
                    var result38 = parse_WS();
                    while (result38 !== null) {
                        result36.push(result38);
                        var result38 = parse_WS();
                    }
                    if (result36 !== null) {
                        var result37 = parse_MultiplicativeExpression();
                        if (result37 !== null) {
                            var result33 = [result34, result35, result36, result37];
                        } else {
                            var result33 = null;
                            pos = savedPos6;
                        }
                    } else {
                        var result33 = null;
                        pos = savedPos6;
```

```
            }
        } else {
            var result33 = null;
            pos = savedPos6;
        }
    } else {
        var result33 = null;
        pos = savedPos6;
    }
    if (result33 !== null) {
        var result5 = result33;
    } else {
        var savedPos5 = pos;
        var result27 = [];
        var result32 = parse_WS();
        while (result32 !== null) {
            result27.push(result32);
            var result32 = parse_WS();
        }
        if (result27 !== null) {
            if (input.substr(pos, 1) === "-") {
                var result28 = "-";
                pos += 1;
            } else {
                var result28 = null;
                if (reportMatchFailures) {
                    matchFailed("\"-\"");
                }
            }
            if (result28 !== null) {
                var result29 = [];
                var result31 = parse_WS();
                while (result31 !== null) {
                    result29.push(result31);
                    var result31 = parse_WS();
                }
                if (result29 !== null) {
                    var result30 = parse_MultiplicativeExpression();
                    if (result30 !== null) {
                        var result26 = [result27, result28, result29, result30];
                    } else {
                        var result26 = null;
                        pos = savedPos5;
                    }
                } else {
                    var result26 = null;
                    pos = savedPos5;
                }
            } else {
                var result26 = null;
                pos = savedPos5;
            }
        } else {
            var result26 = null;
            pos = savedPos5;
        }
        if (result26 !== null) {
            var result5 = result26;
        } else {
            var savedPos2 = pos;
            var result25 = parse_NumericLiteralNegative();
            if (result25 !== null) {
                var result7 = result25;
            } else {
                var result24 = parse_NumericLiteralNegative();
                if (result24 !== null) {
                    var result7 = result24;
```

```
                                    } else {
                                        var result7 = null;;
                                    };
                                }
                                if (result7 !== null) {
                                    var savedPos4 = pos;
                                    var result18 = [];
                                    var result23 = parse_WS();
                                    while (result23 !== null) {
                                        result18.push(result23);
                                        var result23 = parse_WS();
                                    }
                                    if (result18 !== null) {
                                        if (input.substr(pos, 1) === "*") {
                                            var result19 = "*";
                                            pos += 1;
                                        } else {
                                            var result19 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\"*\"");
                                            }
                                        }
                                        if (result19 !== null) {
                                            var result20 = [];
                                            var result22 = parse_WS();
                                            while (result22 !== null) {
                                                result20.push(result22);
                                                var result22 = parse_WS();
                                            }
                                            if (result20 !== null) {
                                                var result21 = parse_UnaryExpression();
                                                if (result21 !== null) {
                                                    var result17 = [result18, result19, result20,
result21];
                                                } else {
                                                    var result17 = null;
                                                    pos = savedPos4;
                                                }
                                            } else {
                                                var result17 = null;
                                                pos = savedPos4;
                                            }
                                        } else {
                                            var result17 = null;
                                            pos = savedPos4;
                                        }
                                    } else {
                                        var result17 = null;
                                        pos = savedPos4;
                                    }
                                    if (result17 !== null) {
                                        var result9 = result17;
                                    } else {
                                        var savedPos3 = pos;
                                        var result11 = [];
                                        var result16 = parse_WS();
                                        while (result16 !== null) {
                                            result11.push(result16);
                                            var result16 = parse_WS();
                                        }
                                        if (result11 !== null) {
                                            if (input.substr(pos, 1) === "/") {
                                                var result12 = "/";
                                                pos += 1;
                                            } else {
                                                var result12 = null;
                                                if (reportMatchFailures) {
```

```
                                        matchFailed("\"/\"");
                                    }
                                }
                                if (result12 !== null) {
                                    var result13 = [];
                                    var result15 = parse_WS();
                                    while (result15 !== null) {
                                        result13.push(result15);
                                        var result15 = parse_WS();
                                    }
                                    if (result13 !== null) {
                                        var result14 = parse_UnaryExpression();
                                        if (result14 !== null) {
                                            var result10 = [result11, result12, result13,
result14];

                                        } else {
                                            var result10 = null;
                                            pos = savedPos3;
                                        }
                                    } else {
                                        var result10 = null;
                                        pos = savedPos3;
                                    }
                                } else {
                                    var result10 = null;
                                    pos = savedPos3;
                                }
                            } else {
                                var result10 = null;
                                pos = savedPos3;
                            }
                            if (result10 !== null) {
                                var result9 = result10;
                            } else {
                                var result9 = null;;
                            };
                        }
                        var result8 = result9 !== null ? result9 : '';
                        if (result8 !== null) {
                            var result6 = [result7, result8];
                        } else {
                            var result6 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result6 = null;
                        pos = savedPos2;
                    }
                    if (result6 !== null) {
                        var result5 = result6;
                    } else {
                        var result5 = null;;
                    };
                };
            }
            while (result5 !== null) {
                result4.push(result5);
                var savedPos6 = pos;
                var result34 = [];
                var result39 = parse_WS();
                while (result39 !== null) {
                    result34.push(result39);
                    var result39 = parse_WS();
                }
                if (result34 !== null) {
                    if (input.substr(pos, 1) === "+") {
                        var result35 = "+";
```

```
                pos += 1;
            } else {
                var result35 = null;
                if (reportMatchFailures) {
                    matchFailed("\"+\"");
                }
            }
            if (result35 !== null) {
                var result36 = [];
                var result38 = parse_WS();
                while (result38 !== null) {
                    result36.push(result38);
                    var result38 = parse_WS();
                }
                if (result36 !== null) {
                    var result37 = parse_MultiplicativeExpression();
                    if (result37 !== null) {
                        var result33 = [result34, result35, result36, result37];
                    } else {
                        var result33 = null;
                        pos = savedPos6;
                    }
                } else {
                    var result33 = null;
                    pos = savedPos6;
                }
            } else {
                var result33 = null;
                pos = savedPos6;
            }
        } else {
            var result33 = null;
            pos = savedPos6;
        }
        if (result33 !== null) {
            var result5 = result33;
        } else {
            var savedPos5 = pos;
            var result27 = [];
            var result32 = parse_WS();
            while (result32 !== null) {
                result27.push(result32);
                var result32 = parse_WS();
            }
            if (result27 !== null) {
                if (input.substr(pos, 1) === "-") {
                    var result28 = "-";
                    pos += 1;
                } else {
                    var result28 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"-\"");
                    }
                }
                if (result28 !== null) {
                    var result29 = [];
                    var result31 = parse_WS();
                    while (result31 !== null) {
                        result29.push(result31);
                        var result31 = parse_WS();
                    }
                    if (result29 !== null) {
                        var result30 = parse_MultiplicativeExpression();
                        if (result30 !== null) {
                            var result26 = [result27, result28, result29, result30];
                        } else {
                            var result26 = null;
```

```
                                    pos = savedPos5;
                                }
                            } else {
                                var result26 = null;
                                pos = savedPos5;
                            }
                        } else {
                            var result26 = null;
                            pos = savedPos5;
                        }
                    } else {
                        var result26 = null;
                        pos = savedPos5;
                    }
                    if (result26 !== null) {
                        var result5 = result26;
                    } else {
                        var savedPos2 = pos;
                        var result25 = parse_NumericLiteralNegative();
                        if (result25 !== null) {
                            var result7 = result25;
                        } else {
                            var result24 = parse_NumericLiteralNegative();
                            if (result24 !== null) {
                                var result7 = result24;
                            } else {
                                var result7 = null;;
                            };
                        }
                        if (result7 !== null) {
                            var savedPos4 = pos;
                            var result18 = [];
                            var result23 = parse_WS();
                            while (result23 !== null) {
                                result18.push(result23);
                                var result23 = parse_WS();
                            }
                            if (result18 !== null) {
                                if (input.substr(pos, 1) === "*") {
                                    var result19 = "*";
                                    pos += 1;
                                } else {
                                    var result19 = null;
                                    if (reportMatchFailures) {
                                        matchFailed("\"*\"");
                                    }
                                }
                                if (result19 !== null) {
                                    var result20 = [];
                                    var result22 = parse_WS();
                                    while (result22 !== null) {
                                        result20.push(result22);
                                        var result22 = parse_WS();
                                    }
                                    if (result20 !== null) {
                                        var result21 = parse_UnaryExpression();
                                        if (result21 !== null) {
                                            var result17 = [result18, result19, result20,
result21];
                                        } else {
                                            var result17 = null;
                                            pos = savedPos4;
                                        }
                                    } else {
                                        var result17 = null;
                                        pos = savedPos4;
                                    }
```

```
                    } else {
                        var result17 = null;
                        pos = savedPos4;
                    }
                } else {
                    var result17 = null;
                    pos = savedPos4;
                }
                if (result17 !== null) {
                    var result9 = result17;
                } else {
                    var savedPos3 = pos;
                    var result11 = [];
                    var result16 = parse_WS();
                    while (result16 !== null) {
                        result11.push(result16);
                        var result16 = parse_WS();
                    }
                    if (result11 !== null) {
                        if (input.substr(pos, 1) === "/") {
                            var result12 = "/";
                            pos += 1;
                        } else {
                            var result12 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"/\"");
                            }
                        }
                        if (result12 !== null) {
                            var result13 = [];
                            var result15 = parse_WS();
                            while (result15 !== null) {
                                result13.push(result15);
                                var result15 = parse_WS();
                            }
                            if (result13 !== null) {
                                var result14 = parse_UnaryExpression();
                                if (result14 !== null) {
                                    var result10 = [result11, result12, result13,
result14];

                                } else {
                                    var result10 = null;
                                    pos = savedPos3;
                                }
                            } else {
                                var result10 = null;
                                pos = savedPos3;
                            }
                        } else {
                            var result10 = null;
                            pos = savedPos3;
                        }
                    } else {
                        var result10 = null;
                        pos = savedPos3;
                    }
                    if (result10 !== null) {
                        var result9 = result10;
                    } else {
                        var result9 = null;;
                    };
                }
                var result8 = result9 !== null ? result9 : '';
                if (result8 !== null) {
                    var result6 = [result7, result8];
                } else {
                    var result6 = null;
```

```
                            pos = savedPos2;
                        }
                    } else {
                        var result6 = null;
                        pos = savedPos2;
                    }
                    if (result6 !== null) {
                        var result5 = result6;
                    } else {
                        var result5 = null;;
                    };
                };
            }
        }
        if (result4 !== null) {
            var result1 = [result3, result4];
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(op1, ops) {
        if(ops.length === 0) {
            return op1;
        }

        var ex = {};
        ex.token = 'expression';
        ex.expressionType = 'additiveexpression';
        ex.summand = op1;
        ex.summands = [];

        for(var i=0; i<ops.length; i++) {
            var summand = ops[i];
            var sum = {};
            if(summand.length == 4 && typeof(summand[1]) === "string") {
                sum.operator = summand[1];
                sum.expression = summand[3];
            } else {
                var subexp = {}
                var firstFactor = sum[0];
                var operator = sum[1][1];
                var secondFactor = sum[1][3];
                var operator = null;
                if(firstFactor.value < 0) {
                    sum.operator = '-';
                    firstFactor.value = - firstFactor.value;
                } else {
                    sum.operator = '+';
                }
                subexp.token = 'expression';
                subexp.expressionType = 'multiplicativeexpression';
                subexp.operator = firstFactor;
                subexp.factors = [{operator: operator, expression: secondFactor}];

                sum.expression = subexp;
            }
            ex.summands.push(sum);
        }

        return ex;
    })(result1[0], result1[1])
        : null;
```

```
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[101] AdditiveExpression");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_MultiplicativeExpression() {
        var cacheKey = 'MultiplicativeExpression@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_UnaryExpression();
        if (result3 !== null) {
            var result4 = [];
            var savedPos3 = pos;
            var result14 = [];
            var result19 = parse_WS();
            while (result19 !== null) {
                result14.push(result19);
                var result19 = parse_WS();
            }
            if (result14 !== null) {
                if (input.substr(pos, 1) === "*") {
                    var result15 = "*";
                    pos += 1;
                } else {
                    var result15 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"*\"");
                    }
                }
                if (result15 !== null) {
                    var result16 = [];
                    var result18 = parse_WS();
                    while (result18 !== null) {
                        result16.push(result18);
                        var result18 = parse_WS();
                    }
                    if (result16 !== null) {
                        var result17 = parse_UnaryExpression();
                        if (result17 !== null) {
                            var result13 = [result14, result15, result16, result17];
                        } else {
                            var result13 = null;
                            pos = savedPos3;
                        }
                    } else {
                        var result13 = null;
```

```
                pos = savedPos3;
            }
        } else {
            var result13 = null;
            pos = savedPos3;
        }
    } else {
        var result13 = null;
        pos = savedPos3;
    }
    if (result13 !== null) {
        var result5 = result13;
    } else {
        var savedPos2 = pos;
        var result7 = [];
        var result12 = parse_WS();
        while (result12 !== null) {
            result7.push(result12);
            var result12 = parse_WS();
        }
        if (result7 !== null) {
            if (input.substr(pos, 1) === "/") {
                var result8 = "/";
                pos += 1;
            } else {
                var result8 = null;
                if (reportMatchFailures) {
                    matchFailed("\"/\"");
                }
            }
            if (result8 !== null) {
                var result9 = [];
                var result11 = parse_WS();
                while (result11 !== null) {
                    result9.push(result11);
                    var result11 = parse_WS();
                }
                if (result9 !== null) {
                    var result10 = parse_UnaryExpression();
                    if (result10 !== null) {
                        var result6 = [result7, result8, result9, result10];
                    } else {
                        var result6 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result6 = null;
                    pos = savedPos2;
                }
            } else {
                var result6 = null;
                pos = savedPos2;
            }
        } else {
            var result6 = null;
            pos = savedPos2;
        }
        if (result6 !== null) {
            var result5 = result6;
        } else {
            var result5 = null;;
        };
    }
    while (result5 !== null) {
        result4.push(result5);
        var savedPos3 = pos;
        var result14 = [];
```

```
            var result19 = parse_WS();
            while (result19 !== null) {
                result14.push(result19);
                var result19 = parse_WS();
            }
            if (result14 !== null) {
                if (input.substr(pos, 1) === "*") {
                    var result15 = "*";
                    pos += 1;
                } else {
                    var result15 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"*\"");
                    }
                }
                if (result15 !== null) {
                    var result16 = [];
                    var result18 = parse_WS();
                    while (result18 !== null) {
                        result16.push(result18);
                        var result18 = parse_WS();
                    }
                    if (result16 !== null) {
                        var result17 = parse_UnaryExpression();
                        if (result17 !== null) {
                            var result13 = [result14, result15, result16, result17];
                        } else {
                            var result13 = null;
                            pos = savedPos3;
                        }
                    } else {
                        var result13 = null;
                        pos = savedPos3;
                    }
                } else {
                    var result13 = null;
                    pos = savedPos3;
                }
            } else {
                var result13 = null;
                pos = savedPos3;
            }
            if (result13 !== null) {
                var result5 = result13;
            } else {
                var savedPos2 = pos;
                var result7 = [];
                var result12 = parse_WS();
                while (result12 !== null) {
                    result7.push(result12);
                    var result12 = parse_WS();
                }
                if (result7 !== null) {
                    if (input.substr(pos, 1) === "/") {
                        var result8 = "/";
                        pos += 1;
                    } else {
                        var result8 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"/\"");
                        }
                    }
                    if (result8 !== null) {
                        var result9 = [];
                        var result11 = parse_WS();
                        while (result11 !== null) {
                            result9.push(result11);
```

```
                        var result11 = parse_WS();
                    }
                    if (result9 !== null) {
                        var result10 = parse_UnaryExpression();
                        if (result10 !== null) {
                            var result6 = [result7, result8, result9, result10];
                        } else {
                            var result6 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result6 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result6 = null;
                    pos = savedPos2;
                }
            } else {
                var result6 = null;
                pos = savedPos2;
            }
            if (result6 !== null) {
                var result5 = result6;
            } else {
                var result5 = null;;
            };
        }
    }
    if (result4 !== null) {
        var result1 = [result3, result4];
    } else {
        var result1 = null;
        pos = savedPos1;
    }
} else {
    var result1 = null;
    pos = savedPos1;
}
var result2 = result1 !== null
    ? (function(exp, exps) {
    if(exps.length === 0) {
        return exp;
    }

    var ex = {};
    ex.token = 'expression';
    ex.expressionType = 'multiplicativeexpression';
    ex.factor = exp;
    ex.factors = [];
    for(var i=0; i<exps.length; i++) {
        var factor = exps[i];
        var fact = {};
        fact.operator = factor[1];
        fact.expression = factor[3];
        ex.factors.push(fact);
    }

    return ex;
})(result1[0], result1[1])
    : null;
if (result2 !== null) {
    var result0 = result2;
} else {
    var result0 = null;
    pos = savedPos0;
}
```

```
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[102] MultiplicativeExpression");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_UnaryExpression() {
        var cacheKey = 'UnaryExpression@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos4 = pos;
        var savedPos5 = pos;
        if (input.substr(pos, 1) === "!") {
            var result19 = "!";
            pos += 1;
        } else {
            var result19 = null;
            if (reportMatchFailures) {
                matchFailed("\"!\"");
            }
        }
        if (result19 !== null) {
            var result20 = [];
            var result22 = parse_WS();
            while (result22 !== null) {
                result20.push(result22);
                var result22 = parse_WS();
            }
            if (result20 !== null) {
                var result21 = parse_PrimaryExpression();
                if (result21 !== null) {
                    var result17 = [result19, result20, result21];
                } else {
                    var result17 = null;
                    pos = savedPos5;
                }
            } else {
                var result17 = null;
                pos = savedPos5;
            }
        } else {
            var result17 = null;
            pos = savedPos5;
        }
        var result18 = result17 !== null
            ? (function(e) {
            var ex = {};
            ex.token = 'expression';
            ex.expressionType = 'unaryexpression';
            ex.unaryexpression = "!";
            ex.expression = e;

            return ex;
        })(result17[2])
            : null;
```

```
    if (result18 !== null) {
        var result16 = result18;
    } else {
        var result16 = null;
        pos = savedPos4;
    }
    if (result16 !== null) {
        var result0 = result16;
    } else {
        var savedPos2 = pos;
        var savedPos3 = pos;
        if (input.substr(pos, 1) === "+") {
            var result12 = "+";
            pos += 1;
        } else {
            var result12 = null;
            if (reportMatchFailures) {
                matchFailed("\"+\"");
            }
        }
        if (result12 !== null) {
            var result13 = [];
            var result15 = parse_WS();
            while (result15 !== null) {
                result13.push(result15);
                var result15 = parse_WS();
            }
            if (result13 !== null) {
                var result14 = parse_PrimaryExpression();
                if (result14 !== null) {
                    var result10 = [result12, result13, result14];
                } else {
                    var result10 = null;
                    pos = savedPos3;
                }
            } else {
                var result10 = null;
                pos = savedPos3;
            }
        } else {
            var result10 = null;
            pos = savedPos3;
        }
        var result11 = result10 !== null
            ? (function(v) {
            var ex = {};
            ex.token = 'expression';
            ex.expressionType = 'unaryexpression';
            ex.unaryexpression = "+";
            ex.expression = v;

            return ex;
        })(result10[2])
            : null;
        if (result11 !== null) {
            var result9 = result11;
        } else {
            var result9 = null;
            pos = savedPos2;
        }
        if (result9 !== null) {
            var result0 = result9;
        } else {
            var savedPos0 = pos;
            var savedPos1 = pos;
            if (input.substr(pos, 1) === "-") {
                var result5 = "-";
```

```
                pos += 1;
            } else {
                var result5 = null;
                if (reportMatchFailures) {
                    matchFailed("\"-\"");
                }
            }
            if (result5 !== null) {
                var result6 = [];
                var result8 = parse_WS();
                while (result8 !== null) {
                    result6.push(result8);
                    var result8 = parse_WS();
                }
                if (result6 !== null) {
                    var result7 = parse_PrimaryExpression();
                    if (result7 !== null) {
                        var result3 = [result5, result6, result7];
                    } else {
                        var result3 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result3 = null;
                    pos = savedPos1;
                }
            } else {
                var result3 = null;
                pos = savedPos1;
            }
            var result4 = result3 !== null
                ? (function(v) {
                var ex = {};
                ex.token = 'expression';
                ex.expressionType = 'unaryexpression';
                ex.unaryexpression = "-";
                ex.expression = v;

                return ex;
            })(result3[2])
                : null;
            if (result4 !== null) {
                var result2 = result4;
            } else {
                var result2 = null;
                pos = savedPos0;
            }
            if (result2 !== null) {
                var result0 = result2;
            } else {
                var result1 = parse_PrimaryExpression();
                if (result1 !== null) {
                    var result0 = result1;
                } else {
                    var result0 = null;;
                };
            };
        };
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[103] UnaryExpression");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
```

```
        };
        return result0;
    }

    function parse_PrimaryExpression() {
        var cacheKey = 'PrimaryExpression@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result16 = parse_BrackettedExpression();
        if (result16 !== null) {
            var result0 = result16;
        } else {
            var result15 = parse_BuiltInCall();
            if (result15 !== null) {
                var result0 = result15;
            } else {
                var result14 = parse_IRIrefOrFunction();
                if (result14 !== null) {
                    var result0 = result14;
                } else {
                    var savedPos3 = pos;
                    var result12 = parse_RDFLiteral();
                    var result13 = result12 !== null
                        ? (function(v) {
                        var ex = {};
                        ex.token = 'expression';
                        ex.expressionType = 'atomic';
                        ex.primaryexpression = 'rdfliteral';
                        ex.value = v;

                        return ex;
                    })(result12)
                        : null;
                    if (result13 !== null) {
                        var result11 = result13;
                    } else {
                        var result11 = null;
                        pos = savedPos3;
                    }
                    if (result11 !== null) {
                        var result0 = result11;
                    } else {
                        var savedPos2 = pos;
                        var result9 = parse_NumericLiteral();
                        var result10 = result9 !== null
                            ? (function(v) {
                            var ex = {};
                            ex.token = 'expression';
                            ex.expressionType = 'atomic';
                            ex.primaryexpression = 'numericliteral';
                            ex.value = v;

                            return ex;
                        })(result9)
                            : null;
                        if (result10 !== null) {
                            var result8 = result10;
                        } else {
                            var result8 = null;
                            pos = savedPos2;
                        }
```

```
                    if (result8 !== null) {
                        var result0 = result8;
                    } else {
                        var savedPos1 = pos;
                        var result6 = parse_BooleanLiteral();
                        var result7 = result6 !== null
                            ? (function(v) {
                            var ex = {};
                            ex.token = 'expression';
                            ex.expressionType = 'atomic';
                            ex.primaryexpression = 'booleanliteral';
                            ex.value = v;

                            return ex;
                        })(result6)
                            : null;
                        if (result7 !== null) {
                            var result5 = result7;
                        } else {
                            var result5 = null;
                            pos = savedPos1;
                        }
                        if (result5 !== null) {
                            var result0 = result5;
                        } else {
                            var result4 = parse_Aggregate();
                            if (result4 !== null) {
                                var result0 = result4;
                            } else {
                                var savedPos0 = pos;
                                var result2 = parse_Var();
                                var result3 = result2 !== null
                                    ? (function(v) {
                                    var ex = {};
                                    ex.token = 'expression';
                                    ex.expressionType = 'atomic';
                                    ex.primaryexpression = 'var';
                                    ex.value = v;

                                    return ex;
                                })(result2)
                                    : null;
                                if (result3 !== null) {
                                    var result1 = result3;
                                } else {
                                    var result1 = null;
                                    pos = savedPos0;
                                }
                                if (result1 !== null) {
                                    var result0 = result1;
                                } else {
                                    var result0 = null;;
                                };
                            };
                        };
                    };
                };
            };
        };
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[104] PrimaryExpression");
    }

    cache[cacheKey] = {
        nextPos: pos,
```

```
            result:  result0
        };
        return result0;
    }

    function parse_BracketedExpression() {
        var cacheKey = 'BracketedExpression@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "(") {
            var result3 = "(";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"(\"");
            }
        }
        if (result3 !== null) {
            var result4 = [];
            var result9 = parse_WS();
            while (result9 !== null) {
                result4.push(result9);
                var result9 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = parse_ConditionalOrExpression();
                if (result5 !== null) {
                    var result6 = [];
                    var result8 = parse_WS();
                    while (result8 !== null) {
                        result6.push(result8);
                        var result8 = parse_WS();
                    }
                    if (result6 !== null) {
                        if (input.substr(pos, 1) === ")") {
                            var result7 = ")";
                            pos += 1;
                        } else {
                            var result7 = null;
                            if (reportMatchFailures) {
                                matchFailed("\")\"");
                            }
                        }
                        if (result7 !== null) {
                            var result1 = [result3, result4, result5, result6, result7];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
```

```
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(e) {
            return e;
        })(result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[105] BrackettedExpression");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_BuiltInCall() {
        var cacheKey = 'BuiltInCall@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos27 = pos;
        var savedPos28 = pos;
        if (input.substr(pos, 3) === "STR") {
            var result216 = "STR";
            pos += 3;
        } else {
            var result216 = null;
            if (reportMatchFailures) {
                matchFailed("\"STR\"");
            }
        }
        if (result216 !== null) {
            var result205 = result216;
        } else {
            if (input.substr(pos, 3) === "str") {
                var result215 = "str";
                pos += 3;
            } else {
                var result215 = null;
                if (reportMatchFailures) {
                    matchFailed("\"str\"");
                }
            }
            if (result215 !== null) {
                var result205 = result215;
            } else {
                var result205 = null;;
```

```
                    };
                }
                if (result205 !== null) {
                    var result206 = [];
                    var result214 = parse_WS();
                    while (result214 !== null) {
                        result206.push(result214);
                        var result214 = parse_WS();
                    }
                    if (result206 !== null) {
                        if (input.substr(pos, 1) === "(") {
                            var result207 = "(";
                            pos += 1;
                        } else {
                            var result207 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"(\"");
                            }
                        }
                        if (result207 !== null) {
                            var result208 = [];
                            var result213 = parse_WS();
                            while (result213 !== null) {
                                result208.push(result213);
                                var result213 = parse_WS();
                            }
                            if (result208 !== null) {
                                var result209 = parse_ConditionalOrExpression();
                                if (result209 !== null) {
                                    var result210 = [];
                                    var result212 = parse_WS();
                                    while (result212 !== null) {
                                        result210.push(result212);
                                        var result212 = parse_WS();
                                    }
                                    if (result210 !== null) {
                                        if (input.substr(pos, 1) === ")") {
                                            var result211 = ")";
                                            pos += 1;
                                        } else {
                                            var result211 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\")\"");
                                            }
                                        }
                                        if (result211 !== null) {
                                            var result203 = [result205, result206, result207,
result208, result209, result210, result211];
                                        } else {
                                            var result203 = null;
                                            pos = savedPos28;
                                        }
                                    } else {
                                        var result203 = null;
                                        pos = savedPos28;
                                    }
                                } else {
                                    var result203 = null;
                                    pos = savedPos28;
                                }
                            } else {
                                var result203 = null;
                                pos = savedPos28;
                            }
                        } else {
                            var result203 = null;
                            pos = savedPos28;
                        }
```

```
            }
        } else {
            var result203 = null;
            pos = savedPos28;
        }
    } else {
        var result203 = null;
        pos = savedPos28;
    }
    var result204 = result203 !== null
        ? (function(e) {
        var ex = {};
        ex.token = 'expression'
        ex.expressionType = 'builtincall'
        ex.builtincall = 'str'
        ex.args = [e]

        return ex;
    })(result203[4])
        : null;
    if (result204 !== null) {
        var result202 = result204;
    } else {
        var result202 = null;
        pos = savedPos27;
    }
    if (result202 !== null) {
        var result0 = result202;
    } else {
        var savedPos25 = pos;
        var savedPos26 = pos;
        if (input.substr(pos, 4) === "LANG") {
            var result201 = "LANG";
            pos += 4;
        } else {
            var result201 = null;
            if (reportMatchFailures) {
                matchFailed("\"LANG\"");
            }
        }
        if (result201 !== null) {
            var result190 = result201;
        } else {
            if (input.substr(pos, 4) === "lang") {
                var result200 = "lang";
                pos += 4;
            } else {
                var result200 = null;
                if (reportMatchFailures) {
                    matchFailed("\"lang\"");
                }
            }
            if (result200 !== null) {
                var result190 = result200;
            } else {
                var result190 = null;;
            };
        }
        if (result190 !== null) {
            var result191 = [];
            var result199 = parse_WS();
            while (result199 !== null) {
                result191.push(result199);
                var result199 = parse_WS();
            }
            if (result191 !== null) {
                if (input.substr(pos, 1) === "(") {
```

```
                                    var result192 = "(";
                                    pos += 1;
                                } else {
                                    var result192 = null;
                                    if (reportMatchFailures) {
                                        matchFailed("\"(\"");
                                    }
                                }
                                if (result192 !== null) {
                                    var result193 = [];
                                    var result198 = parse_WS();
                                    while (result198 !== null) {
                                        result193.push(result198);
                                        var result198 = parse_WS();
                                    }
                                    if (result193 !== null) {
                                        var result194 = parse_ConditionalOrExpression();
                                        if (result194 !== null) {
                                            var result195 = [];
                                            var result197 = parse_WS();
                                            while (result197 !== null) {
                                                result195.push(result197);
                                                var result197 = parse_WS();
                                            }
                                            if (result195 !== null) {
                                                if (input.substr(pos, 1) === ")") {
                                                    var result196 = ")";
                                                    pos += 1;
                                                } else {
                                                    var result196 = null;
                                                    if (reportMatchFailures) {
                                                        matchFailed("\")\"");
                                                    }
                                                }
                                                if (result196 !== null) {
                                                    var result188 = [result190, result191, result192,
result193, result194, result195, result196];
                                                } else {
                                                    var result188 = null;
                                                    pos = savedPos26;
                                                }
                                            } else {
                                                var result188 = null;
                                                pos = savedPos26;
                                            }
                                        } else {
                                            var result188 = null;
                                            pos = savedPos26;
                                        }
                                    } else {
                                        var result188 = null;
                                        pos = savedPos26;
                                    }
                                } else {
                                    var result188 = null;
                                    pos = savedPos26;
                                }
                            } else {
                                var result188 = null;
                                pos = savedPos26;
                            }
                        } else {
                            var result188 = null;
                            pos = savedPos26;
                        }
                        var result189 = result188 !== null
                            ? (function(e) {
```

```
        var ex = {};
        ex.token = 'expression'
        ex.expressionType = 'builtincall'
        ex.builtincall = 'lang'
        ex.args = [e]

        return ex;
    })(result188[4])
        : null;
    if (result189 !== null) {
        var result187 = result189;
    } else {
        var result187 = null;
        pos = savedPos25;
    }
    if (result187 !== null) {
        var result0 = result187;
    } else {
        var savedPos23 = pos;
        var savedPos24 = pos;
        if (input.substr(pos, 11) === "LANGMATCHES") {
            var result171 = "LANGMATCHES";
            pos += 11;
        } else {
            var result171 = null;
            if (reportMatchFailures) {
                matchFailed("\"LANGMATCHES\"");
            }
        }
        if (result171 !== null) {
            var result172 = [];
            var result186 = parse_WS();
            while (result186 !== null) {
                result172.push(result186);
                var result186 = parse_WS();
            }
            if (result172 !== null) {
                if (input.substr(pos, 1) === "(") {
                    var result173 = "(";
                    pos += 1;
                } else {
                    var result173 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
                }
                if (result173 !== null) {
                    var result174 = [];
                    var result185 = parse_WS();
                    while (result185 !== null) {
                        result174.push(result185);
                        var result185 = parse_WS();
                    }
                    if (result174 !== null) {
                        var result175 = parse_ConditionalOrExpression();
                        if (result175 !== null) {
                            var result176 = [];
                            var result184 = parse_WS();
                            while (result184 !== null) {
                                result176.push(result184);
                                var result184 = parse_WS();
                            }
                            if (result176 !== null) {
                                if (input.substr(pos, 1) === ",") {
                                    var result177 = ",";
                                    pos += 1;
                                } else {
```

```
                                                    var result177 = null;
                                                    if (reportMatchFailures) {
                                                        matchFailed("\",\"");
                                                    }
                                                }
                                                if (result177 !== null) {
                                                    var result178 = [];
                                                    var result183 = parse_WS();
                                                    while (result183 !== null) {
                                                        result178.push(result183);
                                                        var result183 = parse_WS();
                                                    }
                                                    if (result178 !== null) {
                                                        var result179 = parse_ConditionalOrExpression
();
                                                        if (result179 !== null) {
                                                            var result180 = [];
                                                            var result182 = parse_WS();
                                                            while (result182 !== null) {
                                                                result180.push(result182);
                                                                var result182 = parse_WS();
                                                            }
                                                            if (result180 !== null) {
                                                                if (input.substr(pos, 1) === ")") {
                                                                    var result181 = ")";
                                                                    pos += 1;
                                                                } else {
                                                                    var result181 = null;
                                                                    if (reportMatchFailures) {
                                                                        matchFailed("\")\"");
                                                                    }
                                                                }
                                                                if (result181 !== null) {
                                                                    var result169 = [result171,
result172, result173, result174, result175, result176, result177, result178, result179, result180,
result181];
                                                                } else {
                                                                    var result169 = null;
                                                                    pos = savedPos24;
                                                                }
                                                            } else {
                                                                var result169 = null;
                                                                pos = savedPos24;
                                                            }
                                                        } else {
                                                            var result169 = null;
                                                            pos = savedPos24;
                                                        }
                                                    } else {
                                                        var result169 = null;
                                                        pos = savedPos24;
                                                    }
                                                } else {
                                                    var result169 = null;
                                                    pos = savedPos24;
                                                }
                                            } else {
                                                var result169 = null;
                                                pos = savedPos24;
                                            }
                                        } else {
                                            var result169 = null;
                                            pos = savedPos24;
                                        }
                                    } else {
                                        var result169 = null;
                                        pos = savedPos24;
```

```
                }
            } else {
                var result169 = null;
                pos = savedPos24;
            }
        } else {
            var result169 = null;
            pos = savedPos24;
        }
    } else {
        var result169 = null;
        pos = savedPos24;
    }
    var result170 = result169 !== null
        ? (function(e1, e2) {
        var ex = {};
        ex.token = 'expression'
        ex.expressionType = 'builtincall'
        ex.builtincall = 'langmatches'
        ex.args = [e1,e2]

        return ex;
    })(result169[4], result169[8])
        : null;
    if (result170 !== null) {
        var result168 = result170;
    } else {
        var result168 = null;
        pos = savedPos23;
    }
    if (result168 !== null) {
        var result0 = result168;
    } else {
        var savedPos21 = pos;
        var savedPos22 = pos;
        if (input.substr(pos, 8) === "DATATYPE") {
            var result167 = "DATATYPE";
            pos += 8;
        } else {
            var result167 = null;
            if (reportMatchFailures) {
                matchFailed("\"DATATYPE\"");
            }
        }
        if (result167 !== null) {
            var result156 = result167;
        } else {
            if (input.substr(pos, 8) === "datatype") {
                var result166 = "datatype";
                pos += 8;
            } else {
                var result166 = null;
                if (reportMatchFailures) {
                    matchFailed("\"datatype\"");
                }
            }
            if (result166 !== null) {
                var result156 = result166;
            } else {
                var result156 = null;;
            };
        }
        if (result156 !== null) {
            var result157 = [];
            var result165 = parse_WS();
            while (result165 !== null) {
                result157.push(result165);
```

```
                                    var result165 = parse_WS();
                                }
                                if (result157 !== null) {
                                    if (input.substr(pos, 1) === "(") {
                                        var result158 = "(";
                                        pos += 1;
                                    } else {
                                        var result158 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("\"(\"");
                                        }
                                    }
                                    if (result158 !== null) {
                                        var result159 = [];
                                        var result164 = parse_WS();
                                        while (result164 !== null) {
                                            result159.push(result164);
                                            var result164 = parse_WS();
                                        }
                                        if (result159 !== null) {
                                            var result160 = parse_ConditionalOrExpression();
                                            if (result160 !== null) {
                                                var result161 = [];
                                                var result163 = parse_WS();
                                                while (result163 !== null) {
                                                    result161.push(result163);
                                                    var result163 = parse_WS();
                                                }
                                                if (result161 !== null) {
                                                    if (input.substr(pos, 1) === ")") {
                                                        var result162 = ")";
                                                        pos += 1;
                                                    } else {
                                                        var result162 = null;
                                                        if (reportMatchFailures) {
                                                            matchFailed("\")\"");
                                                        }
                                                    }
                                                    if (result162 !== null) {
                                                        var result154 = [result156, result157,
result158, result159, result160, result161, result162];
                                                    } else {
                                                        var result154 = null;
                                                        pos = savedPos22;
                                                    }
                                                } else {
                                                    var result154 = null;
                                                    pos = savedPos22;
                                                }
                                            } else {
                                                var result154 = null;
                                                pos = savedPos22;
                                            }
                                        } else {
                                            var result154 = null;
                                            pos = savedPos22;
                                        }
                                    } else {
                                        var result154 = null;
                                        pos = savedPos22;
                                    }
                                } else {
                                    var result154 = null;
                                    pos = savedPos22;
                                }
                            } else {
                                var result154 = null;
```

```
        pos = savedPos22;
    }
    var result155 = result154 !== null
        ? (function(e) {
        var ex = {};
        ex.token = 'expression'
        ex.expressionType = 'builtincall'
        ex.builtincall = 'datatype'
        ex.args = [e]

        return ex;
    })(result154[4])
        : null;
    if (result155 !== null) {
        var result153 = result155;
    } else {
        var result153 = null;
        pos = savedPos21;
    }
    if (result153 !== null) {
        var result0 = result153;
    } else {
        var savedPos19 = pos;
        var savedPos20 = pos;
        if (input.substr(pos, 5) === "BOUND") {
            var result143 = "BOUND";
            pos += 5;
        } else {
            var result143 = null;
            if (reportMatchFailures) {
                matchFailed("\"BOUND\"");
            }
        }
        if (result143 !== null) {
            var result144 = [];
            var result152 = parse_WS();
            while (result152 !== null) {
                result144.push(result152);
                var result152 = parse_WS();
            }
            if (result144 !== null) {
                if (input.substr(pos, 1) === "(") {
                    var result145 = "(";
                    pos += 1;
                } else {
                    var result145 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
                }
                if (result145 !== null) {
                    var result146 = [];
                    var result151 = parse_WS();
                    while (result151 !== null) {
                        result146.push(result151);
                        var result151 = parse_WS();
                    }
                    if (result146 !== null) {
                        var result147 = parse_Var();
                        if (result147 !== null) {
                            var result148 = [];
                            var result150 = parse_WS();
                            while (result150 !== null) {
                                result148.push(result150);
                                var result150 = parse_WS();
                            }
                            if (result148 !== null) {
```

```
                                                    if (input.substr(pos, 1) === ")") {
                                                        var result149 = ")";
                                                        pos += 1;
                                                    } else {
                                                        var result149 = null;
                                                        if (reportMatchFailures) {
                                                            matchFailed("\")\"");
                                                        }
                                                    }
                                                    if (result149 !== null) {
                                                        var result141 = [result143, result144,
result145, result146, result147, result148, result149];
                                                    } else {
                                                        var result141 = null;
                                                        pos = savedPos20;
                                                    }
                                                } else {
                                                    var result141 = null;
                                                    pos = savedPos20;
                                                }
                                            } else {
                                                var result141 = null;
                                                pos = savedPos20;
                                            }
                                        } else {
                                            var result141 = null;
                                            pos = savedPos20;
                                        }
                                    } else {
                                        var result141 = null;
                                        pos = savedPos20;
                                    }
                                } else {
                                    var result141 = null;
                                    pos = savedPos20;
                                }
                                var result142 = result141 !== null
                                    ? (function(v) {
                                    var ex = {};
                                    ex.token = 'expression'
                                    ex.expressionType = 'builtincall'
                                    ex.builtincall = 'bound'
                                    ex.args = [v]

                                    return ex;
                                })(result141[4])
                                    : null;
                                if (result142 !== null) {
                                    var result140 = result142;
                                } else {
                                    var result140 = null;
                                    pos = savedPos19;
                                }
                                if (result140 !== null) {
                                    var result0 = result140;
                                } else {
                                    var savedPos17 = pos;
                                    var savedPos18 = pos;
                                    if (input.substr(pos, 3) === "IRI") {
                                        var result130 = "IRI";
                                        pos += 3;
                                    } else {
                                        var result130 = null;
```

```
                                        if (reportMatchFailures) {
                                            matchFailed("\"IRI\"");
                                        }
                                    }
                                    if (result130 !== null) {
                                        var result131 = [];
                                        var result139 = parse_WS();
                                        while (result139 !== null) {
                                            result131.push(result139);
                                            var result139 = parse_WS();
                                        }
                                        if (result131 !== null) {
                                            if (input.substr(pos, 1) === "(") {
                                                var result132 = "(";
                                                pos += 1;
                                            } else {
                                                var result132 = null;
                                                if (reportMatchFailures) {
                                                    matchFailed("\"(\"");
                                                }
                                            }
                                            if (result132 !== null) {
                                                var result133 = [];
                                                var result138 = parse_WS();
                                                while (result138 !== null) {
                                                    result133.push(result138);
                                                    var result138 = parse_WS();
                                                }
                                                if (result133 !== null) {
                                                    var result134 = parse_ConditionalOrExpression();
                                                    if (result134 !== null) {
                                                        var result135 = [];
                                                        var result137 = parse_WS();
                                                        while (result137 !== null) {
                                                            result135.push(result137);
                                                            var result137 = parse_WS();
                                                        }
                                                        if (result135 !== null) {
                                                            if (input.substr(pos, 1) === ")") {
                                                                var result136 = ")";
                                                                pos += 1;
                                                            } else {
                                                                var result136 = null;
                                                                if (reportMatchFailures) {
                                                                    matchFailed("\")\"");
                                                                }
                                                            }
                                                            if (result136 !== null) {
                                                                var result128 = [result130,
result131, result132, result133, result134, result135, result136];
                                                            } else {
                                                                var result128 = null;
                                                                pos = savedPos18;
                                                            }
                                                        } else {
                                                            var result128 = null;
                                                            pos = savedPos18;
                                                        }
                                                    } else {
                                                        var result128 = null;
                                                        pos = savedPos18;
                                                    }
                                                } else {
                                                    var result128 = null;
                                                    pos = savedPos18;
                                                }
                                            } else {
```

```
                var result128 = null;
                pos = savedPos18;
            }
        } else {
            var result128 = null;
            pos = savedPos18;
        }
    } else {
        var result128 = null;
        pos = savedPos18;
    }
    var result129 = result128 !== null
        ? (function(e) {
        var ex = {};
        ex.token = 'expression';
        ex.expressionType = 'builtincall';
        ex.builtincall = 'iri'
        ex.args = [e];

        return ex;
    })(result128[4])
        : null;
    if (result129 !== null) {
        var result127 = result129;
    } else {
        var result127 = null;
        pos = savedPos17;
    }
    if (result127 !== null) {
        var result0 = result127;
    } else {
        var savedPos15 = pos;
        var savedPos16 = pos;
        if (input.substr(pos, 3) === "URI") {
            var result117 = "URI";
            pos += 3;
        } else {
            var result117 = null;
            if (reportMatchFailures) {
                matchFailed("\"URI\"");
            }
        }
        if (result117 !== null) {
            var result118 = [];
            var result126 = parse_WS();
            while (result126 !== null) {
                result118.push(result126);
                var result126 = parse_WS();
            }
            if (result118 !== null) {
                if (input.substr(pos, 1) === "(") {
                    var result119 = "(";
                    pos += 1;
                } else {
                    var result119 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
                }
                if (result119 !== null) {
                    var result120 = [];
                    var result125 = parse_WS();
                    while (result125 !== null) {
                        result120.push(result125);
                        var result125 = parse_WS();
                    }
                    if (result120 !== null) {
```

```
                                                    var result121 = parse_ConditionalOrExpression
();
                                                    if (result121 !== null) {
                                                        var result122 = [];
                                                        var result124 = parse_WS();
                                                        while (result124 !== null) {
                                                            result122.push(result124);
                                                            var result124 = parse_WS();
                                                        }
                                                        if (result122 !== null) {
                                                            if (input.substr(pos, 1) === ")") {
                                                                var result123 = ")";
                                                                pos += 1;
                                                            } else {
                                                                var result123 = null;
                                                                if (reportMatchFailures) {
                                                                    matchFailed("\")\"");
                                                                }
                                                            }
                                                            if (result123 !== null) {
                                                                var result115 = [result117,
result118, result119, result120, result121, result122, result123];
                                                            } else {
                                                                var result115 = null;
                                                                pos = savedPos16;
                                                            }
                                                        } else {
                                                            var result115 = null;
                                                            pos = savedPos16;
                                                        }
                                                    } else {
                                                        var result115 = null;
                                                        pos = savedPos16;
                                                    }
                                                } else {
                                                    var result115 = null;
                                                    pos = savedPos16;
                                                }
                                            } else {
                                                var result115 = null;
                                                pos = savedPos16;
                                            }
                                        } else {
                                            var result115 = null;
                                            pos = savedPos16;
                                        }
                                    } else {
                                        var result115 = null;
                                        pos = savedPos16;
                                    }
                                    var result116 = result115 !== null
                                        ? (function(e) {
                                        var ex = {};
                                        ex.token = 'expression';
                                        ex.expressionType = 'builtincall';
                                        ex.builtincall = 'uri'
                                        ex.args = [e];

                                        return ex;
                                    })(result115[4])
                                        : null;
                                    if (result116 !== null) {
                                        var result114 = result116;
                                    } else {
                                        var result114 = null;
                                        pos = savedPos15;
                                    }
```

```
                                        if (result114 !== null) {
                                            var result0 = result114;
                                        } else {
                                            var savedPos12 = pos;
                                            var savedPos13 = pos;
                                            if (input.substr(pos, 5) === "BNODE") {
                                                var result101 = "BNODE";
                                                pos += 5;
                                            } else {
                                                var result101 = null;
                                                if (reportMatchFailures) {
                                                    matchFailed("\"BNODE\"");
                                                }
                                            }
                                            if (result101 !== null) {
                                                var result102 = [];
                                                var result113 = parse_WS();
                                                while (result113 !== null) {
                                                    result102.push(result113);
                                                    var result113 = parse_WS();
                                                }
                                                if (result102 !== null) {
                                                    var savedPos14 = pos;
                                                    if (input.substr(pos, 1) === "(") {
                                                        var result106 = "(";
                                                        pos += 1;
                                                    } else {
                                                        var result106 = null;
                                                        if (reportMatchFailures) {
                                                            matchFailed("\"(\"");
                                                        }
                                                    }
                                                    if (result106 !== null) {
                                                        var result107 = [];
                                                        var result112 = parse_WS();
                                                        while (result112 !== null) {
                                                            result107.push(result112);
                                                            var result112 = parse_WS();
                                                        }
                                                        if (result107 !== null) {
                                                            var result108 =
parse_ConditionalOrExpression();
                                                            if (result108 !== null) {
                                                                var result109 = [];
                                                                var result111 = parse_WS();
                                                                while (result111 !== null) {
                                                                    result109.push(result111);
                                                                    var result111 = parse_WS();
                                                                }
                                                                if (result109 !== null) {
                                                                    if (input.substr(pos, 1) === ")")
{
                                                                        var result110 = ")";
                                                                        pos += 1;
                                                                    } else {
                                                                        var result110 = null;
                                                                        if (reportMatchFailures) {
                                                                            matchFailed("\")\"");
                                                                        }
                                                                    }
                                                                    if (result110 !== null) {
                                                                        var result105 = [result106,
result107, result108, result109, result110];
                                                                    } else {
                                                                        var result105 = null;
                                                                        pos = savedPos14;
                                                                    }
```

```
                                    } else {
                                        var result105 = null;
                                        pos = savedPos14;
                                    }
                                } else {
                                    var result105 = null;
                                    pos = savedPos14;
                                }
                            } else {
                                var result105 = null;
                                pos = savedPos14;
                            }
                        } else {
                            var result105 = null;
                            pos = savedPos14;
                        }
                        if (result105 !== null) {
                            var result103 = result105;
                        } else {
                            var result104 = parse_NIL();
                            if (result104 !== null) {
                                var result103 = result104;
                            } else {
                                var result103 = null;;
                            };
                        }
                        if (result103 !== null) {
                            var result99 = [result101, result102,
result103];

                        } else {
                            var result99 = null;
                            pos = savedPos13;
                        }
                    } else {
                        var result99 = null;
                        pos = savedPos13;
                    }
                } else {
                    var result99 = null;
                    pos = savedPos13;
                }
                var result100 = result99 !== null
                    ? (function(arg) {
                    var ex = {};
                    ex.token = 'expression';
                    ex.expressionType = 'builtincall';
                    ex.builtincall = 'bnode';
                    if(arg.length === 5) {
                        ex.args = [arg[2]];
                    } else {
                        ex.args = null;
                    }

                    return ex;
                })(result99[2])
                    : null;
                if (result100 !== null) {
                    var result98 = result100;
                } else {
                    var result98 = null;
                    pos = savedPos12;
                }
                if (result98 !== null) {
                    var result0 = result98;
                } else {
                    var savedPos10 = pos;
                    var savedPos11 = pos;
```

```
                                    if (input.substr(pos, 8) === "COALESCE") {
                                        var result94 = "COALESCE";
                                        pos += 8;
                                    } else {
                                        var result94 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("\"COALESCE\"");
                                        }
                                    }
                                    if (result94 !== null) {
                                        var result95 = [];
                                        var result97 = parse_WS();
                                        while (result97 !== null) {
                                            result95.push(result97);
                                            var result97 = parse_WS();
                                        }
                                        if (result95 !== null) {
                                            var result96 = parse_ExpressionList();
                                            if (result96 !== null) {
                                                var result92 = [result94, result95,
result96];

                                            } else {
                                                var result92 = null;
                                                pos = savedPos11;
                                            }
                                        } else {
                                            var result92 = null;
                                            pos = savedPos11;
                                        }
                                    } else {
                                        var result92 = null;
                                        pos = savedPos11;
                                    }
                                    var result93 = result92 !== null
                                        ? (function(args) {
                                        var ex = {};
                                        ex.token = 'expression';
                                        ex.expressionType = 'builtincall';
                                        ex.builtincall = 'coalesce';
                                        ex.args = args;

                                        return ex;
                                    })(result92[2])
                                        : null;
                                    if (result93 !== null) {
                                        var result91 = result93;
                                    } else {
                                        var result91 = null;
                                        pos = savedPos10;
                                    }
                                    if (result91 !== null) {
                                        var result0 = result91;
                                    } else {
                                        var savedPos8 = pos;
                                        var savedPos9 = pos;
                                        if (input.substr(pos, 2) === "IF") {
                                            var result69 = "IF";
                                            pos += 2;
                                        } else {
                                            var result69 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\"IF\"");
                                            }
                                        }
                                        if (result69 !== null) {
                                            var result70 = [];
                                            var result90 = parse_WS();
```

```
                                                                    while (result90 !== null) {
                                                                        result70.push(result90);
                                                                        var result90 = parse_WS();
                                                                    }
                                                                    if (result70 !== null) {
                                                                        if (input.substr(pos, 1) === "(") {
                                                                            var result71 = "(";
                                                                            pos += 1;
                                                                        } else {
                                                                            var result71 = null;
                                                                            if (reportMatchFailures) {
                                                                                matchFailed("\"(\"");
                                                                            }
                                                                        }
                                                                        if (result71 !== null) {
                                                                            var result72 = [];
                                                                            var result89 = parse_WS();
                                                                            while (result89 !== null) {
                                                                                result72.push(result89);
                                                                                var result89 = parse_WS();
                                                                            }
                                                                            if (result72 !== null) {
                                                                                var result73 =
parse_ConditionalOrExpression();
                                                                                if (result73 !== null) {
                                                                                    var result74 = [];
                                                                                    var result88 = parse_WS();
                                                                                    while (result88 !== null) {
                                                                                        result74.push(result88);
                                                                                        var result88 = parse_WS();
                                                                                    }
                                                                                    if (result74 !== null) {
                                                                                        if (input.substr(pos, 1)
=== ",") {
                                                                                            var result75 = ",";
                                                                                            pos += 1;
                                                                                        } else {
                                                                                            var result75 = null;
                                                                                            if
(reportMatchFailures) {
                                                                                                matchFailed("\",
\"");
                                                                                            }
                                                                                        }
                                                                                        if (result75 !== null) {
                                                                                            var result76 = [];
                                                                                            var result87 =
parse_WS();
                                                                                            while (result87 !==
null) {
                                                                                                result76.push
(result87);
                                                                                                var result87 =
parse_WS();
                                                                                            }
                                                                                            if (result76 !==
null) {
                                                                                                var result77 =
parse_ConditionalOrExpression();
                                                                                                if (result77 !==
null) {
                                                                                                    var result78
= [];
                                                                                                    var result86
= parse_WS();
                                                                                                    while
(result86 !== null) {
```

```
result78.push(result86);
                                                                    var
result86 = parse_WS();
                                                                }
                                                                if (result78 !
== null) {
                                                                    if
(input.substr(pos, 1) === ",") {
                                                                        var
result79 = ",";
                                                                        pos
+= 1;
                                                                    } else {
                                                                        var
result79 = null;
                                                                        if
(reportMatchFailures) {
matchFailed("\",\"");


                                                                        }
                                                                    }
                                                                    if
(result79 !== null) {
                                                                        var
result80 = [];
                                                                        var
result85 = parse_WS();
                                                                        while
(result85 !== null) {
result80.push(result85);
var result85 = parse_WS();


                                                                        }
                                                                        if
(result80 !== null) {
var result81 = parse_ConditionalOrExpression();
                                                                            if
(result81 !== null) {
var result82 = [];
var result84 = parse_WS();
while (result84 !== null) {
result82.push(result84);
var result84 = parse_WS();
                                                                                }
if (result82 !== null) {
if (input.substr(pos, 1) === ")") {
var result83 = ")";
pos += 1;
else {
var result83 = null;
if (reportMatchFailures) {
```

```
matchFailed("\")\"");



if (result83 !== null) {

var result67 = [result69, result70, result71, result72, result73, result74, result75, result76, result77,
result78, result79, result80, result81, result82, result83];

else {

var result67 = null;

pos = savedPos9;

                                                                                                    }
else {

var result67 = null;

pos = savedPos9;
                                                                                                    }
                                                                                                        }
else {

var result67 = null;

pos = savedPos9;
                                                                                                        }
                                                                                                    } else
{

var result67 = null;

pos = savedPos9;
                                                                                                        }
                                                                                                    } else {
                                                                                                        var
result67 = null;
                                                                                                        pos =
savedPos9;
                                                                                                    }
                                                                                                } else {
                                                                                                    var
result67 = null;
                                                                                                    pos =
savedPos9;
                                                                                                }
                                                                                            } else {
                                                                                                var result67
= null;
                                                                                                pos =
savedPos9;
                                                                                            }
                                                                                        } else {
                                                                                            var result67 =
null;
                                                                                            pos = savedPos9;
                                                                                        }
                                                                                    } else {
                                                                                        var result67 = null;
                                                                                        pos = savedPos9;
                                                                                    }
                                                                                } else {
                                                                                    var result67 = null;
                                                                                    pos = savedPos9;
                                                                                }
```

```
                    } else {
                        var result67 = null;
                        pos = savedPos9;
                    }
                } else {
                    var result67 = null;
                    pos = savedPos9;
                }
            } else {
                var result67 = null;
                pos = savedPos9;
            }
        } else {
            var result67 = null;
            pos = savedPos9;
        }
    } else {
        var result67 = null;
        pos = savedPos9;
    }
    var result68 = result67 !== null
        ? (function(test, trueCond, falseCond) {
        var ex = {};
        ex.token = 'expression';
        ex.expressionType = 'builtincall';
        ex.builtincall = 'if';
        ex.args = [test,trueCond,falseCond];

        return ex;
    })(result67[4], result67[8], result67[12])
        : null;
    if (result68 !== null) {
        var result66 = result68;
    } else {
        var result66 = null;
        pos = savedPos8;
    }
    if (result66 !== null) {
        var result0 = result66;
    } else {
        var savedPos6 = pos;
        var savedPos7 = pos;
        if (input.substr(pos, 9) === "ISLITERAL") {
            var result56 = "ISLITERAL";
            pos += 9;
        } else {
            var result56 = null;
            if (reportMatchFailures) {
                matchFailed("\"ISLITERAL\"");
            }
        }
        if (result56 !== null) {
            var result57 = [];
            var result65 = parse_WS();
            while (result65 !== null) {
                result57.push(result65);
                var result65 = parse_WS();
            }
            if (result57 !== null) {
                if (input.substr(pos, 1) === "(") {
                    var result58 = "(";
                    pos += 1;
                } else {
                    var result58 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
```

```
                                                                    }
                                                                    if (result58 !== null) {
                                                                        var result59 = [];
                                                                        var result64 = parse_WS();
                                                                        while (result64 !== null) {
                                                                            result59.push(result64);
                                                                            var result64 = parse_WS();
                                                                        }
                                                                        if (result59 !== null) {
                                                                            var result60 =
parse_ConditionalOrExpression();
                                                                            if (result60 !== null) {
                                                                                var result61 = [];
                                                                                var result63 = parse_WS();
                                                                                while (result63 !== null)
{
                                                                                    result61.push
(result63);
                                                                                    var result63 =
parse_WS();
                                                                                }
                                                                                if (result61 !== null) {
                                                                                    if (input.substr(pos,
1) === ")") {
                                                                                        var result62 =
")";
                                                                                        pos += 1;
                                                                                    } else {
                                                                                        var result62 =
null;
                                                                                        if
(reportMatchFailures) {
                                                                                            matchFailed
("\")\"");
                                                                                        }
                                                                                    }
                                                                                    if (result62 !==
null) {
                                                                                        var result54 =
[result56, result57, result58, result59, result60, result61, result62];
                                                                                    } else {
                                                                                        var result54 =
null;
                                                                                        pos = savedPos7;
                                                                                    }
                                                                                } else {
                                                                                    var result54 = null;
                                                                                    pos = savedPos7;
                                                                                }
                                                                            } else {
                                                                                var result54 = null;
                                                                                pos = savedPos7;
                                                                            }
                                                                        } else {
                                                                            var result54 = null;
                                                                            pos = savedPos7;
                                                                        }
                                                                    } else {
                                                                        var result54 = null;
                                                                        pos = savedPos7;
                                                                    }
                                                                } else {
                                                                    var result54 = null;
```

```
                                                                    pos = savedPos7;
                                                                }
                                                                var result55 = result54 !== null
                                                                    ? (function(arg) {
                                                                    var ex = {};
                                                                    ex.token = 'expression';
                                                                    ex.expressionType = 'builtincall';
                                                                    ex.builtincall = 'isliteral';
                                                                    ex.args = [arg];

                                                                    return ex;
                                                                })(result54[4])
                                                                    : null;
                                                                if (result55 !== null) {
                                                                    var result53 = result55;
                                                                } else {
                                                                    var result53 = null;
                                                                    pos = savedPos6;
                                                                }
                                                                if (result53 !== null) {
                                                                    var result0 = result53;
                                                                } else {
                                                                    var savedPos4 = pos;
                                                                    var savedPos5 = pos;
                                                                    if (input.substr(pos, 7) === "ISBLANK") {
                                                                        var result43 = "ISBLANK";
                                                                        pos += 7;
                                                                    } else {
                                                                        var result43 = null;
                                                                        if (reportMatchFailures) {
                                                                            matchFailed("\"ISBLANK\"");
                                                                        }
                                                                    }
                                                                    if (result43 !== null) {
                                                                        var result44 = [];
                                                                        var result52 = parse_WS();
                                                                        while (result52 !== null) {
                                                                            result44.push(result52);
                                                                            var result52 = parse_WS();
                                                                        }
                                                                        if (result44 !== null) {
                                                                            if (input.substr(pos, 1) === "(")
{

                                                                                var result45 = "(";
                                                                                pos += 1;
                                                                            } else {
                                                                                var result45 = null;
                                                                                if (reportMatchFailures) {
                                                                                    matchFailed("\"(\"");
                                                                                }
                                                                            }
                                                                            if (result45 !== null) {
                                                                                var result46 = [];
                                                                                var result51 = parse_WS();
                                                                                while (result51 !== null) {
                                                                                    result46.push(result51);
                                                                                    var result51 = parse_WS();
                                                                                }
                                                                                if (result46 !== null) {
                                                                                    var result47 =
parse_ConditionalOrExpression();

                                                                                    if (result47 !== null) {
                                                                                        var result48 = [];
                                                                                        var result50 =
parse_WS();

                                                                                        while (result50 !==
null) {
```

```
                                                                    result48.push
(result50);
                                                                    var result50 =
parse_WS();
                                                                }
                                                                if (result48 !==
null) {
                                                                    if (input.substr
(pos, 1) === ")") {
                                                                        var result49
= ")";
                                                                        pos += 1;
                                                                    } else {
                                                                        var result49
= null;
                                                                        if
(reportMatchFailures) {
matchFailed("\")\"");
                                                                        }
                                                                    }
                                                                    if (result49 !==
null) {
                                                                        var result41
= [result43, result44, result45, result46, result47, result48, result49];
                                                                    } else {
                                                                        var result41
= null;
                                                                        pos =
savedPos5;
                                                                    }
                                                                } else {
                                                                    var result41 =
null;
                                                                    pos = savedPos5;
                                                                }
                                                            } else {
                                                                var result41 = null;
                                                                pos = savedPos5;
                                                            }
                                                        } else {
                                                            var result41 = null;
                                                            pos = savedPos5;
                                                        }
                                                    } else {
                                                        var result41 = null;
                                                        pos = savedPos5;
                                                    }
                                                } else {
                                                    var result41 = null;
                                                    pos = savedPos5;
                                                }
                                                var result42 = result41 !== null
                                                    ? (function(arg) {
                                                    var ex = {};
                                                    ex.token = 'expression';
                                                    ex.expressionType = 'builtincall';
                                                    ex.builtincall = 'isblank';
                                                    ex.args = [arg];

                                                    return ex;
                                                })(result41[4])
                                                    : null;
```

```
if (result42 !== null) {
    var result40 = result42;
} else {
    var result40 = null;
    pos = savedPos4;
}
if (result40 !== null) {
    var result0 = result40;
} else {
    var savedPos2 = pos;
    var savedPos3 = pos;
    if (input.substr(pos, 8) ===
"SAMETERM") {
        var result24 = "SAMETERM";
        pos += 8;
    } else {
        var result24 = null;
        if (reportMatchFailures) {
            matchFailed("\"SAMETERM\"");
        }
    }
    if (result24 !== null) {
        var result25 = [];
        var result39 = parse_WS();
        while (result39 !== null) {
            result25.push(result39);
            var result39 = parse_WS();
        }
        if (result25 !== null) {
            if (input.substr(pos, 1) ===
"(") {
                var result26 = "(";
                pos += 1;
            } else {
                var result26 = null;
                if (reportMatchFailures) {
                    matchFailed("\"(\"");
                }
            }
            if (result26 !== null) {
                var result27 = [];
                var result38 = parse_WS();
                while (result38 !== null)
{
                    result27.push
(result38);
                    var result38 =
parse_WS();
                }
                if (result27 !== null) {
                    var result28 =
parse_ConditionalOrExpression();
                    if (result28 !==
null) {
                        var result29 = [];
                        var result37 =
parse_WS();
                        while (result37 !
== null) {
                            result29.push
(result37);
                            var result37
= parse_WS();
                        }
                        if (result29 !==
null) {
                            if
```

```
(input.substr(pos, 1) === ",") {
                                                                    var
result30 = ",";
                                                                    pos += 1;
                                                                } else {
                                                                    var
result30 = null;
                                                                    if
(reportMatchFailures) {
matchFailed("\",\"");
                                                                    }
                                                                }
                                                                if (result30 !
== null) {
                                                                    var
result31 = [];
                                                                    var
result36 = parse_WS();
                                                                    while
(result36 !== null) {
result31.push(result36);
                                                                        var
result36 = parse_WS();
                                                                    }
                                                                    if
(result31 !== null) {
                                                                        var
result32 = parse_ConditionalOrExpression();
                                                                        if
(result32 !== null) {
var result33 = [];
var result35 = parse_WS();
while (result35 !== null) {
result33.push(result35);
var result35 = parse_WS();
                                                                            }
                                                                            if
(result33 !== null) {
if (input.substr(pos, 1) === ")") {
var result34 = ")";
pos += 1;
                                                                                }
else {
var result34 = null;
if (reportMatchFailures) {
matchFailed("\")\"");
                                                                                }
if (result34 !== null) {
var result22 = [result24, result25, result26, result27, result28, result29, result30, result31, result32,
result33, result34];
                                                                                }
```

```
else {
var result22 = null;
pos = savedPos3;
                                                                                                        }
                                                                                                    }
else {
var result22 = null;
pos = savedPos3;
                                                                                                    }
                                                                                                } else
{
var result22 = null;
pos = savedPos3;
                                                                                                }
                                                                                            } else {
                                                                                                var
result22 = null;
                                                                                                pos =
savedPos3;
                                                                                            }
                                                                                        } else {
                                                                                            var
result22 = null;
                                                                                            pos =
savedPos3;
                                                                                        }
                                                                                    } else {
                                                                                        var result22
= null;
                                                                                        pos =
savedPos3;
                                                                                    }
                                                                                } else {
                                                                                    var result22 =
null;
                                                                                    pos = savedPos3;
                                                                                }
                                                                            } else {
                                                                                var result22 = null;
                                                                                pos = savedPos3;
                                                                            }
                                                                        } else {
                                                                            var result22 = null;
                                                                            pos = savedPos3;
                                                                        }
                                                                    } else {
                                                                        var result22 = null;
                                                                        pos = savedPos3;
                                                                    }
                                                                } else {
                                                                    var result22 = null;
                                                                    pos = savedPos3;
                                                                }
                                                                var result23 = result22 !== null
                                                                    ? (function(e1, e2) {
                                                                    var ex = {};
                                                                    ex.token = 'expression';
                                                                    ex.expressionType = 'builtincall';
                                                                    ex.builtincall = 'sameterm';
                                                                    ex.args = [e1, e2];
                                                                    return ex;
```

```
"ISURI") {

"isuri") {

\"");

=== "ISIRI") {

"ISIRI";

(reportMatchFailures) {

("\"ISIRI\"");

result18;

5) === "isiri") {

"isiri";

null;

(reportMatchFailures) {
})(result22[4], result22[8])
    : null;
if (result23 !== null) {
    var result21 = result23;
} else {
    var result21 = null;
    pos = savedPos2;
}
if (result21 !== null) {
    var result0 = result21;
} else {
    var savedPos0 = pos;
    var savedPos1 = pos;
    if (input.substr(pos, 5) ===

        var result20 = "ISURI";
        pos += 5;
    } else {
        var result20 = null;
        if (reportMatchFailures) {
            matchFailed("\"ISURI\"");
        }
    }
    if (result20 !== null) {
        var result7 = result20;
    } else {
        if (input.substr(pos, 5) ===

            var result19 = "isuri";
            pos += 5;
        } else {
            var result19 = null;
            if (reportMatchFailures) {
                matchFailed("\"isuri

            }
        }
        if (result19 !== null) {
            var result7 = result19;
        } else {
            if (input.substr(pos, 5)

                var result18 =

                pos += 5;
            } else {
                var result18 = null;
                if

                    matchFailed

                }
            }
            if (result18 !== null) {
                var result7 =

            } else {
                if (input.substr(pos,

                    var result17 =

                    pos += 5;
                } else {
                    var result17 =

                    if
```

```
("\"isiri\"");


null) {

result17;

null;;


                                                                                matchFailed
                                                                            }
                                                                        }
                                                                        if (result17 !==

                                                                            var result7 =

                                                                        } else {
                                                                            var result7 =

                                                                        };
                                                                    };
                                                                };
                                                            }
                                                            if (result7 !== null) {
                                                                var result8 = [];
                                                                var result16 = parse_WS();
                                                                while (result16 !== null) {
                                                                    result8.push(result16);
                                                                    var result16 = parse_WS();
                                                                }
                                                                if (result8 !== null) {
                                                                    if (input.substr(pos, 1)


=== "(") {


                                                                        var result9 = "(";
                                                                        pos += 1;
                                                                    } else {
                                                                        var result9 = null;
                                                                        if

(reportMatchFailures) {

("\"(\"");


                                                                            matchFailed

                                                                        }
                                                                    }
                                                                    if (result9 !== null) {
                                                                        var result10 = [];
                                                                        var result15 =

parse_WS();

null) {

(result15);

parse_WS();


                                                                        while (result15 !==

null) {

parse_ConditionalOrExpression();

                                                                            result10.push

                                                                            var result15 =

null) {

= [];

                                                                        }
                                                                        if (result10 !==

= parse_WS();

                                                                            var result11 =

(result14 !== null) {

                                                                            if (result11 !==

result12.push(result14);

                                                                                var result12

result14 = parse_WS();

                                                                                var result14


                                                                                while


== null) {

(input.substr(pos, 1) === ")") {


                                                                                    var

                                                                                }
                                                                                if (result12 !

                                                                                    if
```

```
                                                                                                    var
result13 = ")";
                                                                                                    pos
+= 1;
                                                                                                } else {
                                                                                                    var
result13 = null;
                                                                                                    if
(reportMatchFailures) {
matchFailed("\")\"");
                                                                                                    }
                                                                                                }
                                                                                                if
(result13 !== null) {
                                                                                                    var
result5 = [result7, result8, result9, result10, result11, result12, result13];
                                                                                                } else {
                                                                                                    var
result5 = null;
                                                                                                    pos =
savedPos1;
                                                                                                }
                                                                                            } else {
                                                                                                var
result5 = null;
                                                                                                pos =
savedPos1;
                                                                                            }
                                                                                        } else {
                                                                                            var result5 =
null;
                                                                                            pos =
savedPos1;
                                                                                        }
                                                                                    } else {
                                                                                        var result5 =
null;
                                                                                        pos = savedPos1;
                                                                                    }
                                                                                } else {
                                                                                    var result5 = null;
                                                                                    pos = savedPos1;
                                                                                }
                                                                            } else {
                                                                                var result5 = null;
                                                                                pos = savedPos1;
                                                                            }
                                                                        } else {
                                                                            var result5 = null;
                                                                            pos = savedPos1;
                                                                        }
                                                                        var result6 = result5 !== null
                                                                            ? (function(arg) {
                                                                            var ex = {};
                                                                            ex.token = 'expression';
                                                                            ex.expressionType =
'builtincall';
                                                                            ex.builtincall = 'isuri';
                                                                            ex.args = [arg];

                                                                            return ex;
                                                                        })(result5[4])
                                                                            : null;
                                                                        if (result6 !== null) {
                                                                            var result4 = result6;
                                                                        } else {
```

```
                                                                    var result4 = null;
                                                                    pos = savedPos0;
                                                                }
                                                                if (result4 !== null) {
                                                                    var result0 = result4;
                                                                } else {
                                                                    var result3 =
parse_RegexExpression();
                                                                    if (result3 !== null) {
                                                                        var result0 = result3;
                                                                    } else {
                                                                        var result2 =
parse_ExistsFunc();
                                                                        if (result2 !== null) {
                                                                            var result0 = result2;
                                                                        } else {
                                                                            var result1 =
parse_NotExistsFunc();
                                                                            if (result1 !== null)
{
                                                                                var result0 =
result1;
                                                                            } else {
                                                                                var result0 =
null;;
                                                                            };
                                                                        };
                                                                    };
                                                                };
                                                            };
                                                        };
                                                    };
                                                };
                                            };
                                        };
                                    };
                                };
                            };
                        }
                        reportMatchFailures = savedReportMatchFailures;
                        if (reportMatchFailures && result0 === null) {
                            matchFailed("[106] BuiltInCall");
                        }

                        cache[cacheKey] = {
                            nextPos: pos,
                            result:  result0
                        };
                        return result0;
                    }

                    function parse_RegexExpression() {
                        var cacheKey = 'RegexExpression@' + pos;
                        var cachedResult = cache[cacheKey];
                        if (cachedResult) {
                            pos = cachedResult.nextPos;
                            return cachedResult.result;
                        }

                        var savedReportMatchFailures = reportMatchFailures;
                        reportMatchFailures = false;
                        var savedPos0 = pos;
                        var savedPos1 = pos;
                        if (input.substr(pos, 5) === "REGEX") {
```

```
            var result28 = "REGEX";
            pos += 5;
        } else {
            var result28 = null;
            if (reportMatchFailures) {
                matchFailed("\"REGEX\"");
            }
        }
        if (result28 !== null) {
            var result3 = result28;
        } else {
            if (input.substr(pos, 5) === "regex") {
                var result27 = "regex";
                pos += 5;
            } else {
                var result27 = null;
                if (reportMatchFailures) {
                    matchFailed("\"regex\"");
                }
            }
            if (result27 !== null) {
                var result3 = result27;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result26 = parse_WS();
            while (result26 !== null) {
                result4.push(result26);
                var result26 = parse_WS();
            }
            if (result4 !== null) {
                if (input.substr(pos, 1) === "(") {
                    var result5 = "(";
                    pos += 1;
                } else {
                    var result5 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
                }
                if (result5 !== null) {
                    var result6 = [];
                    var result25 = parse_WS();
                    while (result25 !== null) {
                        result6.push(result25);
                        var result25 = parse_WS();
                    }
                    if (result6 !== null) {
                        var result7 = parse_ConditionalOrExpression();
                        if (result7 !== null) {
                            var result8 = [];
                            var result24 = parse_WS();
                            while (result24 !== null) {
                                result8.push(result24);
                                var result24 = parse_WS();
                            }
                            if (result8 !== null) {
                                if (input.substr(pos, 1) === ",") {
                                    var result9 = ",";
                                    pos += 1;
                                } else {
                                    var result9 = null;
                                    if (reportMatchFailures) {
                                        matchFailed("\",\"");
```

```
                                        }
                                    }
                                    if (result9 !== null) {
                                        var result10 = [];
                                        var result23 = parse_WS();
                                        while (result23 !== null) {
                                            result10.push(result23);
                                            var result23 = parse_WS();
                                        }
                                        if (result10 !== null) {
                                            var result11 = parse_ConditionalOrExpression();
                                            if (result11 !== null) {
                                                var result12 = [];
                                                var result22 = parse_WS();
                                                while (result22 !== null) {
                                                    result12.push(result22);
                                                    var result22 = parse_WS();
                                                }
                                                if (result12 !== null) {
                                                    var savedPos2 = pos;
                                                    if (input.substr(pos, 1) === ",") {
                                                        var result18 = ",";
                                                        pos += 1;
                                                    } else {
                                                        var result18 = null;
                                                        if (reportMatchFailures) {
                                                            matchFailed("\",\"");
                                                        }
                                                    }
                                                    if (result18 !== null) {
                                                        var result19 = [];
                                                        var result21 = parse_WS();
                                                        while (result21 !== null) {
                                                            result19.push(result21);
                                                            var result21 = parse_WS();
                                                        }
                                                        if (result19 !== null) {
                                                            var result20 =
parse_ConditionalOrExpression();

                                                            if (result20 !== null) {
                                                                var result17 = [result18,
result19, result20];

                                                            } else {
                                                                var result17 = null;
                                                                pos = savedPos2;
                                                            }
                                                        } else {
                                                            var result17 = null;
                                                            pos = savedPos2;
                                                        }
                                                    } else {
                                                        var result17 = null;
                                                        pos = savedPos2;
                                                    }
                                                    var result13 = result17 !== null ? result17 :
'';

                                                    if (result13 !== null) {
                                                        var result14 = [];
                                                        var result16 = parse_WS();
                                                        while (result16 !== null) {
                                                            result14.push(result16);
                                                            var result16 = parse_WS();
                                                        }
                                                        if (result14 !== null) {
                                                            if (input.substr(pos, 1) === ")") {
                                                                var result15 = ")";
                                                                pos += 1;
```

```
                                                                    } else {
                                                                        var result15 = null;
                                                                        if (reportMatchFailures) {
                                                                            matchFailed("\")\"");
                                                                        }
                                                                    }
                                                                    if (result15 !== null) {
                                                                        var result1 = [result3, result4,
result5, result6, result7, result8, result9, result10, result11, result12, result13, result14, result15];
                                                                    } else {
                                                                        var result1 = null;
                                                                        pos = savedPos1;
                                                                    }
                                                                } else {
                                                                    var result1 = null;
                                                                    pos = savedPos1;
                                                                }
                                                            } else {
                                                                var result1 = null;
                                                                pos = savedPos1;
                                                            }
                                                        } else {
                                                            var result1 = null;
                                                            pos = savedPos1;
                                                        }
                                                    } else {
                                                        var result1 = null;
                                                        pos = savedPos1;
                                                    }
                                                } else {
                                                    var result1 = null;
                                                    pos = savedPos1;
                                                }
                                            } else {
                                                var result1 = null;
                                                pos = savedPos1;
                                            }
                                        } else {
                                            var result1 = null;
                                            pos = savedPos1;
                                        }
                                    } else {
                                        var result1 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result1 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result1 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function(e1, e2, eo) {
                    var regex = {};
                    regex.token = 'expression';
                    regex.expressionType = 'regex';
                    regex.text = e1;
```

```
            regex.pattern = e2;
            regex.flags = eo[2];

            return regex;
        })(result1[4], result1[8], result1[10])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[107] RegexExpression");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ExistsFunc() {
        var cacheKey = 'ExistsFunc@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 6) === "EXISTS") {
            var result8 = "EXISTS";
            pos += 6;
        } else {
            var result8 = null;
            if (reportMatchFailures) {
                matchFailed("\"EXISTS\"");
            }
        }
        if (result8 !== null) {
            var result3 = result8;
        } else {
            if (input.substr(pos, 6) === "exists") {
                var result7 = "exists";
                pos += 6;
            } else {
                var result7 = null;
                if (reportMatchFailures) {
                    matchFailed("\"exists\"");
                }
            }
            if (result7 !== null) {
                var result3 = result7;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result6 = parse_WS();
            while (result6 !== null) {
```

```
                result4.push(result6);
                var result6 = parse_WS();
            }
            if (result4 !== null) {
                var result5 = parse_GroupGraphPattern();
                if (result5 !== null) {
                    var result1 = [result3, result4, result5];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(ggp) {
            var ex = {};
            ex.token = 'expression';
            ex.expressionType = 'builtincall';
            ex.builtincall = 'exists';
            ex.args = [ggp];

            return ex;
        })(result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[108] ExistsFunc");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_NotExistsFunc() {
        var cacheKey = 'NotExistsFunc@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 3) === "NOT") {
            var result13 = "NOT";
            pos += 3;
        } else {
            var result13 = null;
            if (reportMatchFailures) {
                matchFailed("\"NOT\"");
```

```
        }
    }
    if (result13 !== null) {
        var result3 = result13;
    } else {
        if (input.substr(pos, 3) === "not") {
            var result12 = "not";
            pos += 3;
        } else {
            var result12 = null;
            if (reportMatchFailures) {
                matchFailed("\"not\"");
            }
        }
        if (result12 !== null) {
            var result3 = result12;
        } else {
            var result3 = null;;
        };
    }
    if (result3 !== null) {
        var result4 = [];
        var result11 = parse_WS();
        while (result11 !== null) {
            result4.push(result11);
            var result11 = parse_WS();
        }
        if (result4 !== null) {
            if (input.substr(pos, 6) === "EXISTS") {
                var result10 = "EXISTS";
                pos += 6;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("\"EXISTS\"");
                }
            }
            if (result10 !== null) {
                var result5 = result10;
            } else {
                if (input.substr(pos, 6) === "exists") {
                    var result9 = "exists";
                    pos += 6;
                } else {
                    var result9 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"exists\"");
                    }
                }
                if (result9 !== null) {
                    var result5 = result9;
                } else {
                    var result5 = null;;
                };
            }
            if (result5 !== null) {
                var result6 = [];
                var result8 = parse_WS();
                while (result8 !== null) {
                    result6.push(result8);
                    var result8 = parse_WS();
                }
                if (result6 !== null) {
                    var result7 = parse_GroupGraphPattern();
                    if (result7 !== null) {
                        var result1 = [result3, result4, result5, result6, result7];
                    } else {
```

```
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(ggp) {
            var ex = {};
            ex.token = 'expression';
            ex.expressionType = 'builtincall';
            ex.builtincall = 'notexists';
            ex.args = [ggp];

            return ex;
        })(result1[4])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[109] NotExistsFunc");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_Aggregate() {
        var cacheKey = 'Aggregate@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos8 = pos;
        var savedPos9 = pos;
        if (input.substr(pos, 5) === "COUNT") {
            var result117 = "COUNT";
            pos += 5;
        } else {
            var result117 = null;
            if (reportMatchFailures) {
                matchFailed("\"COUNT\"");
```

```
        }
    }
    if (result117 !== null) {
        var result96 = result117;
    } else {
        if (input.substr(pos, 5) === "count") {
            var result116 = "count";
            pos += 5;
        } else {
            var result116 = null;
            if (reportMatchFailures) {
                matchFailed("\"count\"");
            }
        }
        if (result116 !== null) {
            var result96 = result116;
        } else {
            var result96 = null;;
        };
    }
    if (result96 !== null) {
        var result97 = [];
        var result115 = parse_WS();
        while (result115 !== null) {
            result97.push(result115);
            var result115 = parse_WS();
        }
        if (result97 !== null) {
            if (input.substr(pos, 1) === "(") {
                var result98 = "(";
                pos += 1;
            } else {
                var result98 = null;
                if (reportMatchFailures) {
                    matchFailed("\"(\"");
                }
            }
            if (result98 !== null) {
                var result99 = [];
                var result114 = parse_WS();
                while (result114 !== null) {
                    result99.push(result114);
                    var result114 = parse_WS();
                }
                if (result99 !== null) {
                    if (input.substr(pos, 8) === "DISTINCT") {
                        var result113 = "DISTINCT";
                        pos += 8;
                    } else {
                        var result113 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"DISTINCT\"");
                        }
                    }
                    if (result113 !== null) {
                        var result111 = result113;
                    } else {
                        if (input.substr(pos, 8) === "distinct") {
                            var result112 = "distinct";
                            pos += 8;
                        } else {
                            var result112 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"distinct\"");
                            }
                        }
                        if (result112 !== null) {
```

```
                                        var result111 = result112;
                                    } else {
                                        var result111 = null;;
                                    };
                                }
                                var result100 = result111 !== null ? result111 : '';
                                if (result100 !== null) {
                                    var result101 = [];
                                    var result110 = parse_WS();
                                    while (result110 !== null) {
                                        result101.push(result110);
                                        var result110 = parse_WS();
                                    }
                                    if (result101 !== null) {
                                        if (input.substr(pos, 1) === "*") {
                                            var result109 = "*";
                                            pos += 1;
                                        } else {
                                            var result109 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\"*\"");
                                            }
                                        }
                                        if (result109 !== null) {
                                            var result102 = result109;
                                        } else {
                                            var result108 = parse_ConditionalOrExpression();
                                            if (result108 !== null) {
                                                var result102 = result108;
                                            } else {
                                                var result102 = null;;
                                            };
                                        }
                                        if (result102 !== null) {
                                            var result103 = [];
                                            var result107 = parse_WS();
                                            while (result107 !== null) {
                                                result103.push(result107);
                                                var result107 = parse_WS();
                                            }
                                            if (result103 !== null) {
                                                if (input.substr(pos, 1) === ")") {
                                                    var result104 = ")";
                                                    pos += 1;
                                                } else {
                                                    var result104 = null;
                                                    if (reportMatchFailures) {
                                                        matchFailed("\")\"");
                                                    }
                                                }
                                                if (result104 !== null) {
                                                    var result105 = [];
                                                    var result106 = parse_WS();
                                                    while (result106 !== null) {
                                                        result105.push(result106);
                                                        var result106 = parse_WS();
                                                    }
                                                    if (result105 !== null) {
                                                        var result94 = [result96, result97, result98,
result99, result100, result101, result102, result103, result104, result105];
                                                    } else {
                                                        var result94 = null;
                                                        pos = savedPos9;
                                                    }
                                                } else {
                                                    var result94 = null;
                                                    pos = savedPos9;
```

```
                                    }
                                } else {
                                    var result94 = null;
                                    pos = savedPos9;
                                }
                            } else {
                                var result94 = null;
                                pos = savedPos9;
                            }
                        } else {
                            var result94 = null;
                            pos = savedPos9;
                        }
                    } else {
                        var result94 = null;
                        pos = savedPos9;
                    }
                } else {
                    var result94 = null;
                    pos = savedPos9;
                }
            } else {
                var result94 = null;
                pos = savedPos9;
            }
        } else {
            var result94 = null;
            pos = savedPos9;
        }
    } else {
        var result94 = null;
        pos = savedPos9;
    }
    var result95 = result94 !== null
        ? (function(d, e) {
        var exp = {};
        exp.token = 'expression';
        exp.expressionType = 'aggregate';
        exp.aggregateType = 'count';
        exp.distinct = (d != "" ? 'DISTINCT' : d);
        exp.expression = e;

        return exp;

    })(result94[4], result94[6])
        : null;
    if (result95 !== null) {
        var result93 = result95;
    } else {
        var result93 = null;
        pos = savedPos8;
    }
    if (result93 !== null) {
        var result0 = result93;
    } else {
        var savedPos6 = pos;
        var savedPos7 = pos;
        if (input.substr(pos, 3) === "SUM") {
            var result92 = "SUM";
            pos += 3;
        } else {
            var result92 = null;
            if (reportMatchFailures) {
                matchFailed("\"SUM\"");
            }
        }
        if (result92 !== null) {
```

```
            var result73 = result92;
        } else {
            if (input.substr(pos, 3) === "sum") {
                var result91 = "sum";
                pos += 3;
            } else {
                var result91 = null;
                if (reportMatchFailures) {
                    matchFailed("\"sum\"");
                }
            }
            if (result91 !== null) {
                var result73 = result91;
            } else {
                var result73 = null;;
            };
        }
        if (result73 !== null) {
            var result74 = [];
            var result90 = parse_WS();
            while (result90 !== null) {
                result74.push(result90);
                var result90 = parse_WS();
            }
            if (result74 !== null) {
                if (input.substr(pos, 1) === "(") {
                    var result75 = "(";
                    pos += 1;
                } else {
                    var result75 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
                }
                if (result75 !== null) {
                    var result76 = [];
                    var result89 = parse_WS();
                    while (result89 !== null) {
                        result76.push(result89);
                        var result89 = parse_WS();
                    }
                    if (result76 !== null) {
                        if (input.substr(pos, 8) === "DISTINCT") {
                            var result88 = "DISTINCT";
                            pos += 8;
                        } else {
                            var result88 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"DISTINCT\"");
                            }
                        }
                        if (result88 !== null) {
                            var result86 = result88;
                        } else {
                            if (input.substr(pos, 8) === "distinct") {
                                var result87 = "distinct";
                                pos += 8;
                            } else {
                                var result87 = null;
                                if (reportMatchFailures) {
                                    matchFailed("\"distinct\"");
                                }
                            }
                            if (result87 !== null) {
                                var result86 = result87;
                            } else {
                                var result86 = null;;
```

```
                                        };
                                    }
                                    var result77 = result86 !== null ? result86 : '';
                                    if (result77 !== null) {
                                        var result78 = [];
                                        var result85 = parse_WS();
                                        while (result85 !== null) {
                                            result78.push(result85);
                                            var result85 = parse_WS();
                                        }
                                        if (result78 !== null) {
                                            var result79 = parse_ConditionalOrExpression();
                                            if (result79 !== null) {
                                                var result80 = [];
                                                var result84 = parse_WS();
                                                while (result84 !== null) {
                                                    result80.push(result84);
                                                    var result84 = parse_WS();
                                                }
                                                if (result80 !== null) {
                                                    if (input.substr(pos, 1) === ")") {
                                                        var result81 = ")";
                                                        pos += 1;
                                                    } else {
                                                        var result81 = null;
                                                        if (reportMatchFailures) {
                                                            matchFailed("\")\"");
                                                        }
                                                    }
                                                    if (result81 !== null) {
                                                        var result82 = [];
                                                        var result83 = parse_WS();
                                                        while (result83 !== null) {
                                                            result82.push(result83);
                                                            var result83 = parse_WS();
                                                        }
                                                        if (result82 !== null) {
                                                            var result71 = [result73, result74,
result75, result76, result77, result78, result79, result80, result81, result82];
                                                        } else {
                                                            var result71 = null;
                                                            pos = savedPos7;
                                                        }
                                                    } else {
                                                        var result71 = null;
                                                        pos = savedPos7;
                                                    }
                                                } else {
                                                    var result71 = null;
                                                    pos = savedPos7;
                                                }
                                            } else {
                                                var result71 = null;
                                                pos = savedPos7;
                                            }
                                        } else {
                                            var result71 = null;
                                            pos = savedPos7;
                                        }
                                    } else {
                                        var result71 = null;
                                        pos = savedPos7;
                                    }
                                } else {
                                    var result71 = null;
                                    pos = savedPos7;
                                }
```

```
            } else {
                var result71 = null;
                pos = savedPos7;
            }
        } else {
            var result71 = null;
            pos = savedPos7;
        }
    } else {
        var result71 = null;
        pos = savedPos7;
    }
    var result72 = result71 !== null
        ? (function(d, e) {
        var exp = {};
        exp.token = 'expression';
        exp.expressionType = 'aggregate';
        exp.aggregateType = 'sum';
        exp.distinct = (d != "" ? 'DISTINCT' : d);
        exp.expression = e;

        return exp;

    })(result71[4], result71[6])
        : null;
    if (result72 !== null) {
        var result70 = result72;
    } else {
        var result70 = null;
        pos = savedPos6;
    }
    if (result70 !== null) {
        var result0 = result70;
    } else {
        var savedPos4 = pos;
        var savedPos5 = pos;
        if (input.substr(pos, 3) === "MIN") {
            var result69 = "MIN";
            pos += 3;
        } else {
            var result69 = null;
            if (reportMatchFailures) {
                matchFailed("\"MIN\"");
            }
        }
        if (result69 !== null) {
            var result50 = result69;
        } else {
            if (input.substr(pos, 3) === "min") {
                var result68 = "min";
                pos += 3;
            } else {
                var result68 = null;
                if (reportMatchFailures) {
                    matchFailed("\"min\"");
                }
            }
            if (result68 !== null) {
                var result50 = result68;
            } else {
                var result50 = null;;
            };
        }
        if (result50 !== null) {
            var result51 = [];
            var result67 = parse_WS();
            while (result67 !== null) {
```

```
            result51.push(result67);
            var result67 = parse_WS();
        }
        if (result51 !== null) {
            if (input.substr(pos, 1) === "(") {
                var result52 = "(";
                pos += 1;
            } else {
                var result52 = null;
                if (reportMatchFailures) {
                    matchFailed("\"(\"");
                }
            }
            if (result52 !== null) {
                var result53 = [];
                var result66 = parse_WS();
                while (result66 !== null) {
                    result53.push(result66);
                    var result66 = parse_WS();
                }
                if (result53 !== null) {
                    if (input.substr(pos, 8) === "DISTINCT") {
                        var result65 = "DISTINCT";
                        pos += 8;
                    } else {
                        var result65 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"DISTINCT\"");
                        }
                    }
                    if (result65 !== null) {
                        var result63 = result65;
                    } else {
                        if (input.substr(pos, 8) === "distinct") {
                            var result64 = "distinct";
                            pos += 8;
                        } else {
                            var result64 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"distinct\"");
                            }
                        }
                        if (result64 !== null) {
                            var result63 = result64;
                        } else {
                            var result63 = null;;
                        };
                    }
                    var result54 = result63 !== null ? result63 : '';
                    if (result54 !== null) {
                        var result55 = [];
                        var result62 = parse_WS();
                        while (result62 !== null) {
                            result55.push(result62);
                            var result62 = parse_WS();
                        }
                        if (result55 !== null) {
                            var result56 = parse_ConditionalOrExpression();
                            if (result56 !== null) {
                                var result57 = [];
                                var result61 = parse_WS();
                                while (result61 !== null) {
                                    result57.push(result61);
                                    var result61 = parse_WS();
                                }
                                if (result57 !== null) {
                                    if (input.substr(pos, 1) === ")") {
```

```
                                                    var result58 = ")";
                                                    pos += 1;
                                                } else {
                                                    var result58 = null;
                                                    if (reportMatchFailures) {
                                                        matchFailed("\")\"");
                                                    }
                                                }
                                                if (result58 !== null) {
                                                    var result59 = [];
                                                    var result60 = parse_WS();
                                                    while (result60 !== null) {
                                                        result59.push(result60);
                                                        var result60 = parse_WS();
                                                    }
                                                    if (result59 !== null) {
                                                        var result48 = [result50, result51, result52, result53, result54, result55, result56, result57, result58, result59];
                                                    } else {
                                                        var result48 = null;
                                                        pos = savedPos5;
                                                    }
                                                } else {
                                                    var result48 = null;
                                                    pos = savedPos5;
                                                }
                                            } else {
                                                var result48 = null;
                                                pos = savedPos5;
                                            }
                                        } else {
                                            var result48 = null;
                                            pos = savedPos5;
                                        }
                                    } else {
                                        var result48 = null;
                                        pos = savedPos5;
                                    }
                                } else {
                                    var result48 = null;
                                    pos = savedPos5;
                                }
                            } else {
                                var result48 = null;
                                pos = savedPos5;
                            }
                        } else {
                            var result48 = null;
                            pos = savedPos5;
                        }
                    } else {
                        var result48 = null;
                        pos = savedPos5;
                    }
                } else {
                    var result48 = null;
                    pos = savedPos5;
                }
                var result49 = result48 !== null
                    ? (function(d, e) {
                    var exp = {};
                    exp.token = 'expression';
                    exp.expressionType = 'aggregate';
                    exp.aggregateType = 'min';
                    exp.distinct = (d != "" ? 'DISTINCT' : d);
                    exp.expression = e;
```

```
            return exp;

        })(result48[4], result48[6])
            : null;
        if (result49 !== null) {
            var result47 = result49;
        } else {
            var result47 = null;
            pos = savedPos4;
        }
        if (result47 !== null) {
            var result0 = result47;
        } else {
            var savedPos2 = pos;
            var savedPos3 = pos;
            if (input.substr(pos, 3) === "MAX") {
                var result46 = "MAX";
                pos += 3;
            } else {
                var result46 = null;
                if (reportMatchFailures) {
                    matchFailed("\"MAX\"");
                }
            }
            if (result46 !== null) {
                var result27 = result46;
            } else {
                if (input.substr(pos, 3) === "max") {
                    var result45 = "max";
                    pos += 3;
                } else {
                    var result45 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"max\"");
                    }
                }
                if (result45 !== null) {
                    var result27 = result45;
                } else {
                    var result27 = null;;
                };
            }
            if (result27 !== null) {
                var result28 = [];
                var result44 = parse_WS();
                while (result44 !== null) {
                    result28.push(result44);
                    var result44 = parse_WS();
                }
                if (result28 !== null) {
                    if (input.substr(pos, 1) === "(") {
                        var result29 = "(";
                        pos += 1;
                    } else {
                        var result29 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"(\"");
                        }
                    }
                    if (result29 !== null) {
                        var result30 = [];
                        var result43 = parse_WS();
                        while (result43 !== null) {
                            result30.push(result43);
                            var result43 = parse_WS();
                        }
                        if (result30 !== null) {
```

```
                                    if (input.substr(pos, 8) === "DISTINCT") {
                                        var result42 = "DISTINCT";
                                        pos += 8;
                                    } else {
                                        var result42 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("\"DISTINCT\"");
                                        }
                                    }
                                    if (result42 !== null) {
                                        var result40 = result42;
                                    } else {
                                        if (input.substr(pos, 8) === "distinct") {
                                            var result41 = "distinct";
                                            pos += 8;
                                        } else {
                                            var result41 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("\"distinct\"");
                                            }
                                        }
                                        if (result41 !== null) {
                                            var result40 = result41;
                                        } else {
                                            var result40 = null;;
                                        };
                                    }
                                    var result31 = result40 !== null ? result40 : '';
                                    if (result31 !== null) {
                                        var result32 = [];
                                        var result39 = parse_WS();
                                        while (result39 !== null) {
                                            result32.push(result39);
                                            var result39 = parse_WS();
                                        }
                                        if (result32 !== null) {
                                            var result33 = parse_ConditionalOrExpression();
                                            if (result33 !== null) {
                                                var result34 = [];
                                                var result38 = parse_WS();
                                                while (result38 !== null) {
                                                    result34.push(result38);
                                                    var result38 = parse_WS();
                                                }
                                                if (result34 !== null) {
                                                    if (input.substr(pos, 1) === ")") {
                                                        var result35 = ")";
                                                        pos += 1;
                                                    } else {
                                                        var result35 = null;
                                                        if (reportMatchFailures) {
                                                            matchFailed("\")\"");
                                                        }
                                                    }
                                                    if (result35 !== null) {
                                                        var result36 = [];
                                                        var result37 = parse_WS();
                                                        while (result37 !== null) {
                                                            result36.push(result37);
                                                            var result37 = parse_WS();
                                                        }
                                                        if (result36 !== null) {
                                                            var result25 = [result27,
result28, result29, result30, result31, result32, result33, result34, result35, result36];
                                                        } else {
                                                            var result25 = null;
                                                            pos = savedPos3;
```

```
                                        }
                                    } else {
                                        var result25 = null;
                                        pos = savedPos3;
                                    }
                                } else {
                                    var result25 = null;
                                    pos = savedPos3;
                                }
                            } else {
                                var result25 = null;
                                pos = savedPos3;
                            }
                        } else {
                            var result25 = null;
                            pos = savedPos3;
                        }
                    } else {
                        var result25 = null;
                        pos = savedPos3;
                    }
                } else {
                    var result25 = null;
                    pos = savedPos3;
                }
            } else {
                var result25 = null;
                pos = savedPos3;
            }
        } else {
            var result25 = null;
            pos = savedPos3;
        }
    } else {
        var result25 = null;
        pos = savedPos3;
    }
    var result26 = result25 !== null
        ? (function(d, e) {
        var exp = {};
        exp.token = 'expression'
        exp.expressionType = 'aggregate'
        exp.aggregateType = 'max'
        exp.distinct = (d != "" ? 'DISTINCT' : d);
        exp.expression = e

        return exp

    })(result25[4], result25[6])
        : null;
    if (result26 !== null) {
        var result24 = result26;
    } else {
        var result24 = null;
        pos = savedPos2;
    }
    if (result24 !== null) {
        var result0 = result24;
    } else {
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 3) === "AVG") {
            var result23 = "AVG";
            pos += 3;
        } else {
            var result23 = null;
            if (reportMatchFailures) {
```

```
                matchFailed("\"AVG\"");
            }
        }
        if (result23 !== null) {
            var result4 = result23;
        } else {
            if (input.substr(pos, 3) === "avg") {
                var result22 = "avg";
                pos += 3;
            } else {
                var result22 = null;
                if (reportMatchFailures) {
                    matchFailed("\"avg\"");
                }
            }
            if (result22 !== null) {
                var result4 = result22;
            } else {
                var result4 = null;;
            };
        }
        if (result4 !== null) {
            var result5 = [];
            var result21 = parse_WS();
            while (result21 !== null) {
                result5.push(result21);
                var result21 = parse_WS();
            }
            if (result5 !== null) {
                if (input.substr(pos, 1) === "(") {
                    var result6 = "(";
                    pos += 1;
                } else {
                    var result6 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
                }
                if (result6 !== null) {
                    var result7 = [];
                    var result20 = parse_WS();
                    while (result20 !== null) {
                        result7.push(result20);
                        var result20 = parse_WS();
                    }
                    if (result7 !== null) {
                        if (input.substr(pos, 8) === "DISTINCT") {
                            var result19 = "DISTINCT";
                            pos += 8;
                        } else {
                            var result19 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"DISTINCT\"");
                            }
                        }
                        if (result19 !== null) {
                            var result17 = result19;
                        } else {
                            if (input.substr(pos, 8) === "distinct") {
                                var result18 = "distinct";
                                pos += 8;
                            } else {
                                var result18 = null;
                                if (reportMatchFailures) {
                                    matchFailed("\"distinct\"");
                                }
                            }
```

```
                    if (result18 !== null) {
                        var result17 = result18;
                    } else {
                        var result17 = null;;
                    };
                }
                var result8 = result17 !== null ? result17 : '';
                if (result8 !== null) {
                    var result9 = [];
                    var result16 = parse_WS();
                    while (result16 !== null) {
                        result9.push(result16);
                        var result16 = parse_WS();
                    }
                    if (result9 !== null) {
                        var result10 = parse_ConditionalOrExpression
();
                        if (result10 !== null) {
                            var result11 = [];
                            var result15 = parse_WS();
                            while (result15 !== null) {
                                result11.push(result15);
                                var result15 = parse_WS();
                            }
                            if (result11 !== null) {
                                if (input.substr(pos, 1) === ")") {
                                    var result12 = ")";
                                    pos += 1;
                                } else {
                                    var result12 = null;
                                    if (reportMatchFailures) {
                                        matchFailed("\")\"");
                                    }
                                }
                                if (result12 !== null) {
                                    var result13 = [];
                                    var result14 = parse_WS();
                                    while (result14 !== null) {
                                        result13.push(result14);
                                        var result14 = parse_WS();
                                    }
                                    if (result13 !== null) {
                                        var result2 = [result4,
result5, result6, result7, result8, result9, result10, result11, result12, result13];
                                    } else {
                                        var result2 = null;
                                        pos = savedPos1;
                                    }
                                } else {
                                    var result2 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result2 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result2 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result2 = null;
                    pos = savedPos1;
```

```
                                    }
                                } else {
                                    var result2 = null;
                                    pos = savedPos1;
                                }
                            } else {
                                var result2 = null;
                                pos = savedPos1;
                            }
                        } else {
                            var result2 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                    var result3 = result2 !== null
                        ? (function(d, e) {
                        var exp = {};
                        exp.token = 'expression'
                        exp.expressionType = 'aggregate'
                        exp.aggregateType = 'avg'
                        exp.distinct = (d != "" ? 'DISTINCT' : d);
                        exp.expression = e

                        return exp

                    })(result2[4], result2[6])
                        : null;
                    if (result3 !== null) {
                        var result1 = result3;
                    } else {
                        var result1 = null;
                        pos = savedPos0;
                    }
                    if (result1 !== null) {
                        var result0 = result1;
                    } else {
                        var result0 = null;;
                    };
                };
            };
        };
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[110] Aggregate");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_IRIrefOrFunction() {
    var cacheKey = 'IRIrefOrFunction@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
```

```
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_IRIref();
        if (result3 !== null) {
            var result5 = parse_ArgList();
            var result4 = result5 !== null ? result5 : '';
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(i, args) {
            var fcall = {};
            fcall.token = "expression";
            fcall.expressionType = 'irireforfunction';
            fcall.iriref = i;
            fcall.args = args.value;

            return fcall;
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[117] IRIrefOrFunction");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_RDFLiteral() {
        var cacheKey = 'RDFLiteral@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_String();
        if (result3 !== null) {
            var result9 = parse_LANGTAG();
            if (result9 !== null) {
                var result5 = result9;
            } else {
                var savedPos2 = pos;
                if (input.substr(pos, 2) === "^^") {
                    var result7 = "^^";
                    pos += 2;
```

```
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"^^\"");
                    }
                }
                if (result7 !== null) {
                    var result8 = parse_IRIref();
                    if (result8 !== null) {
                        var result6 = [result7, result8];
                    } else {
                        var result6 = null;
                        pos = savedPos2;
                    }
                } else {
                    var result6 = null;
                    pos = savedPos2;
                }
                if (result6 !== null) {
                    var result5 = result6;
                } else {
                    var result5 = null;;
                };
            }
            var result4 = result5 !== null ? result5 : '';
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(s, e) {
            if(typeof(e) === "string" && e.length > 0) {
                return {token:'literal', value:s.value, lang:e.slice(1), type:null}
            } else {
                if(typeof(e) === "object") {
                    e.shift(); // remove the '^^' char
                    return {token:'literal', value:s.value, lang:null, type:e[0] }
                } else {
                    return { token:'literal', value:s.value, lang:null, type:null }
                }
            }
        })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[112] RDFLiteral");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }
```

```
        function parse_NumericLiteral() {
            var cacheKey = 'NumericLiteral@' + pos;
            var cachedResult = cache[cacheKey];
            if (cachedResult) {
                pos = cachedResult.nextPos;
                return cachedResult.result;
            }

            var savedReportMatchFailures = reportMatchFailures;
            reportMatchFailures = false;
            var result3 = parse_NumericLiteralUnsigned();
            if (result3 !== null) {
                var result0 = result3;
            } else {
                var result2 = parse_NumericLiteralPositive();
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result1 = parse_NumericLiteralNegative();
                    if (result1 !== null) {
                        var result0 = result1;
                    } else {
                        var result0 = null;;
                    };
                };
            }
            reportMatchFailures = savedReportMatchFailures;
            if (reportMatchFailures && result0 === null) {
                matchFailed("[113] NumericLiteral");
            }

            cache[cacheKey] = {
                nextPos: pos,
                result:  result0
            };
            return result0;
        }

        function parse_NumericLiteralUnsigned() {
            var cacheKey = 'NumericLiteralUnsigned@' + pos;
            var cachedResult = cache[cacheKey];
            if (cachedResult) {
                pos = cachedResult.nextPos;
                return cachedResult.result;
            }

            var savedReportMatchFailures = reportMatchFailures;
            reportMatchFailures = false;
            var result3 = parse_DOUBLE();
            if (result3 !== null) {
                var result0 = result3;
            } else {
                var result2 = parse_DECIMAL();
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result1 = parse_INTEGER();
                    if (result1 !== null) {
                        var result0 = result1;
                    } else {
                        var result0 = null;;
                    };
                };
            }
            reportMatchFailures = savedReportMatchFailures;
            if (reportMatchFailures && result0 === null) {
                matchFailed("[114] NumericLiteralUnsigned");
```

```
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_NumericLiteralPositive() {
        var cacheKey = 'NumericLiteralPositive@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result3 = parse_DOUBLE_POSITIVE();
        if (result3 !== null) {
            var result0 = result3;
        } else {
            var result2 = parse_DECIMAL_POSITIVE();
            if (result2 !== null) {
                var result0 = result2;
            } else {
                var result1 = parse_INTEGER_POSITIVE();
                if (result1 !== null) {
                    var result0 = result1;
                } else {
                    var result0 = null;;
                };
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[115] NumericLiteralPositive");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_NumericLiteralNegative() {
        var cacheKey = 'NumericLiteralNegative@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result3 = parse_DOUBLE_NEGATIVE();
        if (result3 !== null) {
            var result0 = result3;
        } else {
            var result2 = parse_DECIMAL_NEGATIVE();
            if (result2 !== null) {
                var result0 = result2;
            } else {
                var result1 = parse_INTEGER_NEGATIVE();
                if (result1 !== null) {
```

```
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        };
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[116] NumericLiteralNegative");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
    return result0;
}

function parse_BooleanLiteral() {
    var cacheKey = 'BooleanLiteral@' + pos;
    var cachedResult = cache[cacheKey];
    if (cachedResult) {
        pos = cachedResult.nextPos;
        return cachedResult.result;
    }

    var savedReportMatchFailures = reportMatchFailures;
    reportMatchFailures = false;
    var savedPos1 = pos;
    if (input.substr(pos, 4) === "TRUE") {
        var result10 = "TRUE";
        pos += 4;
    } else {
        var result10 = null;
        if (reportMatchFailures) {
            matchFailed("\"TRUE\"");
        }
    }
    if (result10 !== null) {
        var result7 = result10;
    } else {
        if (input.substr(pos, 4) === "true") {
            var result9 = "true";
            pos += 4;
        } else {
            var result9 = null;
            if (reportMatchFailures) {
                matchFailed("\"true\"");
            }
        }
        if (result9 !== null) {
            var result7 = result9;
        } else {
            var result7 = null;;
        };
    }
    var result8 = result7 !== null
        ? (function() {
        var lit = {};
        lit.token = "literal";
        lit.lang = null;
        lit.type = "http://www.w3.org/2001/XMLSchema#boolean";
        lit.value = true;
        return lit;
    })()
        : null;
    if (result8 !== null) {
```

```
        var result6 = result8;
    } else {
        var result6 = null;
        pos = savedPos1;
    }
    if (result6 !== null) {
        var result0 = result6;
    } else {
        var savedPos0 = pos;
        if (input.substr(pos, 5) === "FALSE") {
            var result5 = "FALSE";
            pos += 5;
        } else {
            var result5 = null;
            if (reportMatchFailures) {
                matchFailed("\"FALSE\"");
            }
        }
        if (result5 !== null) {
            var result2 = result5;
        } else {
            if (input.substr(pos, 5) === "false") {
                var result4 = "false";
                pos += 5;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("\"false\"");
                }
            }
            if (result4 !== null) {
                var result2 = result4;
            } else {
                var result2 = null;;
            };
        }
        var result3 = result2 !== null
            ? (function() {
            var lit = {};
            lit.token = "literal";
            lit.lang = null;
            lit.type = "http://www.w3.org/2001/XMLSchema#boolean";
            lit.value = false;
            return lit;
        })()
            : null;
        if (result3 !== null) {
            var result1 = result3;
        } else {
            var result1 = null;
            pos = savedPos0;
        }
        if (result1 !== null) {
            var result0 = result1;
        } else {
            var result0 = null;;
        };
    }
    reportMatchFailures = savedReportMatchFailures;
    if (reportMatchFailures && result0 === null) {
        matchFailed("[117] BooleanLiteral");
    }

    cache[cacheKey] = {
        nextPos: pos,
        result:  result0
    };
```

```
        return result0;
    }

    function parse_String() {
        var cacheKey = 'String@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos3 = pos;
        var result11 = parse_STRING_LITERAL_LONG1();
        var result12 = result11 !== null
            ? (function(s) { return {token:'string', value:s} })(result11)
            : null;
        if (result12 !== null) {
            var result10 = result12;
        } else {
            var result10 = null;
            pos = savedPos3;
        }
        if (result10 !== null) {
            var result0 = result10;
        } else {
            var savedPos2 = pos;
            var result8 = parse_STRING_LITERAL_LONG2();
            var result9 = result8 !== null
                ? (function(s) { return {token:'string', value:s} })(result8)
                : null;
            if (result9 !== null) {
                var result7 = result9;
            } else {
                var result7 = null;
                pos = savedPos2;
            }
            if (result7 !== null) {
                var result0 = result7;
            } else {
                var savedPos1 = pos;
                var result5 = parse_STRING_LITERAL1();
                var result6 = result5 !== null
                    ? (function(s) { return {token:'string', value:s} })(result5)
                    : null;
                if (result6 !== null) {
                    var result4 = result6;
                } else {
                    var result4 = null;
                    pos = savedPos1;
                }
                if (result4 !== null) {
                    var result0 = result4;
                } else {
                    var savedPos0 = pos;
                    var result2 = parse_STRING_LITERAL2();
                    var result3 = result2 !== null
                        ? (function(s) { return {token:'string', value:s} })(result2)
                        : null;
                    if (result3 !== null) {
                        var result1 = result3;
                    } else {
                        var result1 = null;
                        pos = savedPos0;
                    }
                    if (result1 !== null) {
```

```
                            var result0 = result1;
                        } else {
                            var result0 = null;;
                        };
                    };
                };
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[118] String");
                }

                cache[cacheKey] = {
                    nextPos: pos,
                    result:  result0
                };
                return result0;
            }

            function parse_IRIref() {
                var cacheKey = 'IRIref@' + pos;
                var cachedResult = cache[cacheKey];
                if (cachedResult) {
                    pos = cachedResult.nextPos;
                    return cachedResult.result;
                }

                var savedReportMatchFailures = reportMatchFailures;
                reportMatchFailures = false;
                var savedPos1 = pos;
                var result5 = parse_IRI_REF();
                var result6 = result5 !== null
                    ? (function(iri) { return {token: 'uri', prefix:null, suffix:null, value:iri} })
(result5)
                    : null;
                if (result6 !== null) {
                    var result4 = result6;
                } else {
                    var result4 = null;
                    pos = savedPos1;
                }
                if (result4 !== null) {
                    var result0 = result4;
                } else {
                    var savedPos0 = pos;
                    var result2 = parse_PrefixedName();
                    var result3 = result2 !== null
                        ? (function(p) { return p })(result2)
                        : null;
                    if (result3 !== null) {
                        var result1 = result3;
                    } else {
                        var result1 = null;
                        pos = savedPos0;
                    }
                    if (result1 !== null) {
                        var result0 = result1;
                    } else {
                        var result0 = null;;
                    };
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[119] IRIref");
                }

                cache[cacheKey] = {
```

```
                    nextPos: pos,
                    result:  result0
                };
                return result0;
            }

            function parse_PrefixedName() {
                var cacheKey = 'PrefixedName@' + pos;
                var cachedResult = cache[cacheKey];
                if (cachedResult) {
                    pos = cachedResult.nextPos;
                    return cachedResult.result;
                }

                var savedReportMatchFailures = reportMatchFailures;
                reportMatchFailures = false;
                var savedPos1 = pos;
                var result5 = parse_PNAME_LN();
                var result6 = result5 !== null
                    ? (function(p) { return {token: 'uri', prefix:p[0], suffix:p[1], value:null } })
(result5)
                    : null;
                if (result6 !== null) {
                    var result4 = result6;
                } else {
                    var result4 = null;
                    pos = savedPos1;
                }
                if (result4 !== null) {
                    var result0 = result4;
                } else {
                    var savedPos0 = pos;
                    var result2 = parse_PNAME_NS();
                    var result3 = result2 !== null
                        ? (function(p) { return {token: 'uri', prefix:p, suffix:'', value:null } })
(result2)
                        : null;
                    if (result3 !== null) {
                        var result1 = result3;
                    } else {
                        var result1 = null;
                        pos = savedPos0;
                    }
                    if (result1 !== null) {
                        var result0 = result1;
                    } else {
                        var result0 = null;;
                    };
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[120] PrefixedName");
                }

                cache[cacheKey] = {
                    nextPos: pos,
                    result:  result0
                };
                return result0;
            }

            function parse_BlankNode() {
                var cacheKey = 'BlankNode@' + pos;
                var cachedResult = cache[cacheKey];
                if (cachedResult) {
                    pos = cachedResult.nextPos;
                    return cachedResult.result;
```

```
                    }

                    var savedReportMatchFailures = reportMatchFailures;
                    reportMatchFailures = false;
                    var savedPos1 = pos;
                    var result5 = parse_BLANK_NODE_LABEL();
                    var result6 = result5 !== null
                        ? (function(l) { return {token:'blank', value:l}})(result5)
                        : null;
                    if (result6 !== null) {
                        var result4 = result6;
                    } else {
                        var result4 = null;
                        pos = savedPos1;
                    }
                    if (result4 !== null) {
                        var result0 = result4;
                    } else {
                        var savedPos0 = pos;
                        var result2 = parse_ANON();
                        var result3 = result2 !== null
                            ? (function() { GlobalBlankNodeCounter++; return {token:'blank',
value:'_:'+GlobalBlankNodeCounter} })()
                            : null;
                        if (result3 !== null) {
                            var result1 = result3;
                        } else {
                            var result1 = null;
                            pos = savedPos0;
                        }
                        if (result1 !== null) {
                            var result0 = result1;
                        } else {
                            var result0 = null;;
                        };
                    }
                    reportMatchFailures = savedReportMatchFailures;
                    if (reportMatchFailures && result0 === null) {
                        matchFailed("[121] BlankNode");
                    }

                    cache[cacheKey] = {
                        nextPos: pos,
                        result:  result0
                    };
                    return result0;
                }

                function parse_IRI_REF() {
                    var cacheKey = 'IRI_REF@' + pos;
                    var cachedResult = cache[cacheKey];
                    if (cachedResult) {
                        pos = cachedResult.nextPos;
                        return cachedResult.result;
                    }

                    var savedReportMatchFailures = reportMatchFailures;
                    reportMatchFailures = false;
                    var savedPos0 = pos;
                    var savedPos1 = pos;
                    if (input.substr(pos, 1) === "<") {
                        var result3 = "<";
                        pos += 1;
                    } else {
                        var result3 = null;
                        if (reportMatchFailures) {
                            matchFailed("\"<\"");
```

```
    }
}
if (result3 !== null) {
    var result4 = [];
    if (input.substr(pos).match(/^[^<>"{}|^`\\]/) !== null) {
        var result6 = input.charAt(pos);
        pos++;
    } else {
        var result6 = null;
        if (reportMatchFailures) {
            matchFailed("[^<>\"{}|^`\\\\]");
        }
    }
    while (result6 !== null) {
        result4.push(result6);
        if (input.substr(pos).match(/^[^<>"{}|^`\\]/) !== null) {
            var result6 = input.charAt(pos);
            pos++;
        } else {
            var result6 = null;
            if (reportMatchFailures) {
                matchFailed("[^<>\"{}|^`\\\\]");
            }
        }
    }
    if (result4 !== null) {
        if (input.substr(pos, 1) === ">") {
            var result5 = ">";
            pos += 1;
        } else {
            var result5 = null;
            if (reportMatchFailures) {
                matchFailed("\">\"");
            }
        }
        if (result5 !== null) {
            var result1 = [result3, result4, result5];
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
} else {
    var result1 = null;
    pos = savedPos1;
}
var result2 = result1 !== null
    ? (function(iri_ref) { return iri_ref.join('') })(result1[1])
    : null;
if (result2 !== null) {
    var result0 = result2;
} else {
    var result0 = null;
    pos = savedPos0;
}
reportMatchFailures = savedReportMatchFailures;
if (reportMatchFailures && result0 === null) {
    matchFailed("[122] IRI_REF");
}

cache[cacheKey] = {
    nextPos: pos,
    result:  result0
};
```

```
        return result0;
    }

    function parse_PNAME_NS() {
        var cacheKey = 'PNAME_NS@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result5 = parse_PN_PREFIX();
        var result3 = result5 !== null ? result5 : '';
        if (result3 !== null) {
            if (input.substr(pos, 1) === ":") {
                var result4 = ":";
                pos += 1;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("\":\"");
                }
            }
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(p) { return p })(result1[0])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[123] PNAME_NS");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PNAME_LN() {
        var cacheKey = 'PNAME_LN@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
```

```
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_PNAME_NS();
        if (result3 !== null) {
            var result4 = parse_PN_LOCAL();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(p, s) { return [p, s] })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[124] PNAME_LN");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_BLANK_NODE_LABEL() {
        var cacheKey = 'BLANK_NODE_LABEL@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 2) === "_:") {
            var result3 = "_:";
            pos += 2;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"_:\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_PN_LOCAL();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
```

```
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(l) { return l })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[125] BLANK_NODE_LABEL");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_VAR1() {
        var cacheKey = 'VAR1@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "?") {
            var result3 = "?";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"?\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_VARNAME();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(v) { return v })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
```

```
        if (reportMatchFailures && result0 === null) {
            matchFailed("[126] VAR1");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_VAR2() {
        var cacheKey = 'VAR2@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "$") {
            var result3 = "$";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"$\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_VARNAME();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(v) { return v })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[127] VAR2");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_LANGTAG() {
        var cacheKey = 'LANGTAG@' + pos;
```

```
var cachedResult = cache[cacheKey];
if (cachedResult) {
    pos = cachedResult.nextPos;
    return cachedResult.result;
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
if (input.substr(pos, 1) === "@") {
    var result3 = "@";
    pos += 1;
} else {
    var result3 = null;
    if (reportMatchFailures) {
        matchFailed("\"@\"");
    }
}
if (result3 !== null) {
    if (input.substr(pos).match(/^[a-zA-Z]/) !== null) {
        var result10 = input.charAt(pos);
        pos++;
    } else {
        var result10 = null;
        if (reportMatchFailures) {
            matchFailed("[a-zA-Z]");
        }
    }
    if (result10 !== null) {
        var result4 = [];
        while (result10 !== null) {
            result4.push(result10);
            if (input.substr(pos).match(/^[a-zA-Z]/) !== null) {
                var result10 = input.charAt(pos);
                pos++;
            } else {
                var result10 = null;
                if (reportMatchFailures) {
                    matchFailed("[a-zA-Z]");
                }
            }
        }
    } else {
        var result4 = null;
    }
    if (result4 !== null) {
        var result5 = [];
        var savedPos2 = pos;
        if (input.substr(pos, 1) === "-") {
            var result7 = "-";
            pos += 1;
        } else {
            var result7 = null;
            if (reportMatchFailures) {
                matchFailed("\"-\"");
            }
        }
        if (result7 !== null) {
            if (input.substr(pos).match(/^[a-zA-Z0-9]/) !== null) {
                var result9 = input.charAt(pos);
                pos++;
            } else {
                var result9 = null;
                if (reportMatchFailures) {
                    matchFailed("[a-zA-Z0-9]");
                }
```

```
                }
                if (result9 !== null) {
                    var result8 = [];
                    while (result9 !== null) {
                        result8.push(result9);
                        if (input.substr(pos).match(/^[a-zA-Z0-9]/) !== null) {
                            var result9 = input.charAt(pos);
                            pos++;
                        } else {
                            var result9 = null;
                            if (reportMatchFailures) {
                                matchFailed("[a-zA-Z0-9]");
                            }
                        }
                    }
                } else {
                    var result8 = null;
                }
                if (result8 !== null) {
                    var result6 = [result7, result8];
                } else {
                    var result6 = null;
                    pos = savedPos2;
                }
            } else {
                var result6 = null;
                pos = savedPos2;
            }
            while (result6 !== null) {
                result5.push(result6);
                var savedPos2 = pos;
                if (input.substr(pos, 1) === "-") {
                    var result7 = "-";
                    pos += 1;
                } else {
                    var result7 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"-\"");
                    }
                }
                if (result7 !== null) {
                    if (input.substr(pos).match(/^[a-zA-Z0-9]/) !== null) {
                        var result9 = input.charAt(pos);
                        pos++;
                    } else {
                        var result9 = null;
                        if (reportMatchFailures) {
                            matchFailed("[a-zA-Z0-9]");
                        }
                    }
                    if (result9 !== null) {
                        var result8 = [];
                        while (result9 !== null) {
                            result8.push(result9);
                            if (input.substr(pos).match(/^[a-zA-Z0-9]/) !== null) {
                                var result9 = input.charAt(pos);
                                pos++;
                            } else {
                                var result9 = null;
                                if (reportMatchFailures) {
                                    matchFailed("[a-zA-Z0-9]");
                                }
                            }
                        }
                    } else {
                        var result8 = null;
                    }
```

```
                        if (result8 !== null) {
                            var result6 = [result7, result8];
                        } else {
                            var result6 = null;
                            pos = savedPos2;
                        }
                    } else {
                        var result6 = null;
                        pos = savedPos2;
                    }
                }
                if (result5 !== null) {
                    var result1 = [result3, result4, result5];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(a, b) {

            if(b.length===0) {
                return ("@"+a.join('')).toLowerCase();
            } else {
                return ("@"+a.join('')+"-"+b[0][1].join('')).toLowerCase();
            }
        })(result1[1], result1[2])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[128] LANGTAG");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_INTEGER() {
        var cacheKey = 'INTEGER@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        if (input.substr(pos).match(/^[0-9]/) !== null) {
            var result3 = input.charAt(pos);
            pos++;
```

```
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("[0-9]");
            }
        }
        if (result3 !== null) {
            var result1 = [];
            while (result3 !== null) {
                result1.push(result3);
                if (input.substr(pos).match(/^[0-9]/) !== null) {
                    var result3 = input.charAt(pos);
                    pos++;
                } else {
                    var result3 = null;
                    if (reportMatchFailures) {
                        matchFailed("[0-9]");
                    }
                }
            }
        } else {
            var result1 = null;
        }
        var result2 = result1 !== null
            ? (function(d) {
            var lit = {};
            lit.token = "literal";
            lit.lang = null;
            lit.type = "http://www.w3.org/2001/XMLSchema#integer";
            lit.value = flattenString(d);
            return lit;
        })(result1)
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[129] INTEGER");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DECIMAL() {
        var cacheKey = 'DECIMAL@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos2 = pos;
        var savedPos3 = pos;
        if (input.substr(pos).match(/^[0-9]/) !== null) {
            var result14 = input.charAt(pos);
            pos++;
        } else {
```

```
        var result14 = null;
        if (reportMatchFailures) {
            matchFailed("[0-9]");
        }
    }
    if (result14 !== null) {
        var result10 = [];
        while (result14 !== null) {
            result10.push(result14);
            if (input.substr(pos).match(/^[0-9]/) !== null) {
                var result14 = input.charAt(pos);
                pos++;
            } else {
                var result14 = null;
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
        }
    } else {
        var result10 = null;
    }
    if (result10 !== null) {
        if (input.substr(pos, 1) === ".") {
            var result11 = ".";
            pos += 1;
        } else {
            var result11 = null;
            if (reportMatchFailures) {
                matchFailed("\".\"");
            }
        }
        if (result11 !== null) {
            var result12 = [];
            if (input.substr(pos).match(/^[0-9]/) !== null) {
                var result13 = input.charAt(pos);
                pos++;
            } else {
                var result13 = null;
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
            while (result13 !== null) {
                result12.push(result13);
                if (input.substr(pos).match(/^[0-9]/) !== null) {
                    var result13 = input.charAt(pos);
                    pos++;
                } else {
                    var result13 = null;
                    if (reportMatchFailures) {
                        matchFailed("[0-9]");
                    }
                }
            }
            if (result12 !== null) {
                var result8 = [result10, result11, result12];
            } else {
                var result8 = null;
                pos = savedPos3;
            }
        } else {
            var result8 = null;
            pos = savedPos3;
        }
    } else {
        var result8 = null;
```

```
        pos = savedPos3;
    }
    var result9 = result8 !== null
        ? (function(a, b, c) {

        var lit = {};
        lit.token = "literal";
        lit.lang = null;
        lit.type = "http://www.w3.org/2001/XMLSchema#decimal";
        lit.value = flattenString([a,b,c]);
        return lit;
    })(result8[0], result8[1], result8[2])
        : null;
    if (result9 !== null) {
        var result7 = result9;
    } else {
        var result7 = null;
        pos = savedPos2;
    }
    if (result7 !== null) {
        var result0 = result7;
    } else {
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === ".") {
            var result4 = ".";
            pos += 1;
        } else {
            var result4 = null;
            if (reportMatchFailures) {
                matchFailed("\".\"");
            }
        }
        if (result4 !== null) {
            if (input.substr(pos).match(/^[0-9]/) !== null) {
                var result6 = input.charAt(pos);
                pos++;
            } else {
                var result6 = null;
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
            if (result6 !== null) {
                var result5 = [];
                while (result6 !== null) {
                    result5.push(result6);
                    if (input.substr(pos).match(/^[0-9]/) !== null) {
                        var result6 = input.charAt(pos);
                        pos++;
                    } else {
                        var result6 = null;
                        if (reportMatchFailures) {
                            matchFailed("[0-9]");
                        }
                    }
                }
            } else {
                var result5 = null;
            }
            if (result5 !== null) {
                var result2 = [result4, result5];
            } else {
                var result2 = null;
                pos = savedPos1;
            }
        } else {
```

```
                var result2 = null;
                pos = savedPos1;
            }
            var result3 = result2 !== null
                ? (function(a, b) {
                var lit = {};
                lit.token = "literal";
                lit.lang = null;
                lit.type = "http://www.w3.org/2001/XMLSchema#decimal";
                lit.value = flattenString([a,b]);
                return lit;
            })(result2[0], result2[1])
                : null;
            if (result3 !== null) {
                var result1 = result3;
            } else {
                var result1 = null;
                pos = savedPos0;
            }
            if (result1 !== null) {
                var result0 = result1;
            } else {
                var result0 = null;;
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[130] DECIMAL");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DOUBLE() {
        var cacheKey = 'DOUBLE@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos4 = pos;
        var savedPos5 = pos;
        if (input.substr(pos).match(/^[0-9]/) !== null) {
            var result22 = input.charAt(pos);
            pos++;
        } else {
            var result22 = null;
            if (reportMatchFailures) {
                matchFailed("[0-9]");
            }
        }
        if (result22 !== null) {
            var result17 = [];
            while (result22 !== null) {
                result17.push(result22);
                if (input.substr(pos).match(/^[0-9]/) !== null) {
                    var result22 = input.charAt(pos);
                    pos++;
                } else {
                    var result22 = null;
```

```
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
        }
    } else {
        var result17 = null;
    }
    if (result17 !== null) {
        if (input.substr(pos, 1) === ".") {
            var result18 = ".";
            pos += 1;
        } else {
            var result18 = null;
            if (reportMatchFailures) {
                matchFailed("\".\"");
            }
        }
        if (result18 !== null) {
            var result19 = [];
            if (input.substr(pos).match(/^[0-9]/) !== null) {
                var result21 = input.charAt(pos);
                pos++;
            } else {
                var result21 = null;
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
            while (result21 !== null) {
                result19.push(result21);
                if (input.substr(pos).match(/^[0-9]/) !== null) {
                    var result21 = input.charAt(pos);
                    pos++;
                } else {
                    var result21 = null;
                    if (reportMatchFailures) {
                        matchFailed("[0-9]");
                    }
                }
            }
            if (result19 !== null) {
                var result20 = parse_EXPONENT();
                if (result20 !== null) {
                    var result15 = [result17, result18, result19, result20];
                } else {
                    var result15 = null;
                    pos = savedPos5;
                }
            } else {
                var result15 = null;
                pos = savedPos5;
            }
        } else {
            var result15 = null;
            pos = savedPos5;
        }
    } else {
        var result15 = null;
        pos = savedPos5;
    }
    var result16 = result15 !== null
        ? (function(a, b, c, e) {
        var lit = {};
        lit.token = "literal";
        lit.lang = null;
        lit.type = "http://www.w3.org/2001/XMLSchema#double";
```

```
        lit.value = flattenString([a,b,c,e]);
        return lit;
    })(result15[0], result15[1], result15[2], result15[3])
        : null;
    if (result16 !== null) {
        var result14 = result16;
    } else {
        var result14 = null;
        pos = savedPos4;
    }
    if (result14 !== null) {
        var result0 = result14;
    } else {
        var savedPos2 = pos;
        var savedPos3 = pos;
        if (input.substr(pos, 1) === ".") {
            var result10 = ".";
            pos += 1;
        } else {
            var result10 = null;
            if (reportMatchFailures) {
                matchFailed("\".\"");
            }
        }
        if (result10 !== null) {
            if (input.substr(pos).match(/^[0-9]/) !== null) {
                var result13 = input.charAt(pos);
                pos++;
            } else {
                var result13 = null;
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
            if (result13 !== null) {
                var result11 = [];
                while (result13 !== null) {
                    result11.push(result13);
                    if (input.substr(pos).match(/^[0-9]/) !== null) {
                        var result13 = input.charAt(pos);
                        pos++;
                    } else {
                        var result13 = null;
                        if (reportMatchFailures) {
                            matchFailed("[0-9]");
                        }
                    }
                }
            } else {
                var result11 = null;
            }
            if (result11 !== null) {
                var result12 = parse_EXPONENT();
                if (result12 !== null) {
                    var result8 = [result10, result11, result12];
                } else {
                    var result8 = null;
                    pos = savedPos3;
                }
            } else {
                var result8 = null;
                pos = savedPos3;
            }
        } else {
            var result8 = null;
            pos = savedPos3;
        }
```

```
            var result9 = result8 !== null
                ? (function(a, b, c) {
                var lit = {};
                lit.token = "literal";
                lit.lang = null;
                lit.type = "http://www.w3.org/2001/XMLSchema#double";
                lit.value = flattenString([a,b,c]);
                return lit;
            })(result8[0], result8[1], result8[2])
                : null;
            if (result9 !== null) {
                var result7 = result9;
            } else {
                var result7 = null;
                pos = savedPos2;
            }
            if (result7 !== null) {
                var result0 = result7;
            } else {
                var savedPos0 = pos;
                var savedPos1 = pos;
                if (input.substr(pos).match(/^[0-9]/) !== null) {
                    var result6 = input.charAt(pos);
                    pos++;
                } else {
                    var result6 = null;
                    if (reportMatchFailures) {
                        matchFailed("[0-9]");
                    }
                }
                if (result6 !== null) {
                    var result4 = [];
                    while (result6 !== null) {
                        result4.push(result6);
                        if (input.substr(pos).match(/^[0-9]/) !== null) {
                            var result6 = input.charAt(pos);
                            pos++;
                        } else {
                            var result6 = null;
                            if (reportMatchFailures) {
                                matchFailed("[0-9]");
                            }
                        }
                    }
                } else {
                    var result4 = null;
                }
                if (result4 !== null) {
                    var result5 = parse_EXPONENT();
                    if (result5 !== null) {
                        var result2 = [result4, result5];
                    } else {
                        var result2 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result2 = null;
                    pos = savedPos1;
                }
                var result3 = result2 !== null
                    ? (function(a, b) {
                    var lit = {};
                    lit.token = "literal";
                    lit.lang = null;
                    lit.type = "http://www.w3.org/2001/XMLSchema#double";
                    lit.value = flattenString([a,b]);
                    return lit;
```

```
                })(result2[0], result2[1])
                    : null;
                if (result3 !== null) {
                    var result1 = result3;
                } else {
                    var result1 = null;
                    pos = savedPos0;
                }
                if (result1 !== null) {
                    var result0 = result1;
                } else {
                    var result0 = null;;
                };
            };
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[131] DOUBLE");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_INTEGER_POSITIVE() {
        var cacheKey = 'INTEGER_POSITIVE@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "+") {
            var result3 = "+";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"+\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_INTEGER();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(d) { d.value = "+"+d.value; return d; })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
```

```
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[132] INTEGER_POSITIVE");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DECIMAL_POSITIVE() {
        var cacheKey = 'DECIMAL_POSITIVE@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "+") {
            var result3 = "+";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"+\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_DECIMAL();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(d) { d.value = "+"+d.value; return d })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[133] DECIMAL_POSITIVE");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }
```

```
    function parse_DOUBLE_POSITIVE() {
        var cacheKey = 'DOUBLE_POSITIVE@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "+") {
            var result3 = "+";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"+\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_DOUBLE();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(d) { d.value = "+"+d.value; return d })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[134] DOUBLE_POSITIVE");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_INTEGER_NEGATIVE() {
        var cacheKey = 'INTEGER_NEGATIVE@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
```

```
        if (input.substr(pos, 1) === "-") {
            var result3 = "-";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"-\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_INTEGER();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(d) { d.value = "-"+d.value; return d; })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[135] INTEGER_NEGATIVE");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DECIMAL_NEGATIVE() {
        var cacheKey = 'DECIMAL_NEGATIVE@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "-") {
            var result3 = "-";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"-\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_DECIMAL();
            if (result4 !== null) {
                var result1 = [result3, result4];
```

```
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(d) { d.value = "-"+d.value; return d; })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[136] DECIMAL_NEGATIVE");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_DOUBLE_NEGATIVE() {
        var cacheKey = 'DOUBLE_NEGATIVE@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "-") {
            var result3 = "-";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"-\"");
            }
        }
        if (result3 !== null) {
            var result4 = parse_DOUBLE();
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(d) { d.value = "-"+d.value; return d; })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
```

```
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[137] DOUBLE_NEGATIVE");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_EXPONENT() {
        var cacheKey = 'EXPONENT@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos).match(/^[eE]/) !== null) {
            var result3 = input.charAt(pos);
            pos++;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("[eE]");
            }
        }
        if (result3 !== null) {
            if (input.substr(pos).match(/^[+\-]/) !== null) {
                var result7 = input.charAt(pos);
                pos++;
            } else {
                var result7 = null;
                if (reportMatchFailures) {
                    matchFailed("[+\\-]");
                }
            }
            var result4 = result7 !== null ? result7 : '';
            if (result4 !== null) {
                if (input.substr(pos).match(/^[0-9]/) !== null) {
                    var result6 = input.charAt(pos);
                    pos++;
                } else {
                    var result6 = null;
                    if (reportMatchFailures) {
                        matchFailed("[0-9]");
                    }
                }
                if (result6 !== null) {
                    var result5 = [];
                    while (result6 !== null) {
                        result5.push(result6);
                        if (input.substr(pos).match(/^[0-9]/) !== null) {
                            var result6 = input.charAt(pos);
                            pos++;
                        } else {
                            var result6 = null;
```

```
                                    if (reportMatchFailures) {
                                        matchFailed("[0-9]");
                                    }
                                }
                            }
                        } else {
                            var result5 = null;
                        }
                        if (result5 !== null) {
                            var result1 = [result3, result4, result5];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function(a, b, c) { return flattenString([a,b,c]) })(result1[0], result1[1],
result1[2])
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[138] EXPONENT");
                }

                cache[cacheKey] = {
                    nextPos: pos,
                    result:  result0
                };
                return result0;
            }

            function parse_STRING_LITERAL1() {
                var cacheKey = 'STRING_LITERAL1@' + pos;
                var cachedResult = cache[cacheKey];
                if (cachedResult) {
                    pos = cachedResult.nextPos;
                    return cachedResult.result;
                }

                var savedReportMatchFailures = reportMatchFailures;
                reportMatchFailures = false;
                var savedPos0 = pos;
                var savedPos1 = pos;
                if (input.substr(pos, 1) === "'") {
                    var result3 = "'";
                    pos += 1;
                } else {
                    var result3 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"'\"");
                    }
                }
                if (result3 !== null) {
                    var result4 = [];
```

```
        if (input.substr(pos).match(/^[^'\\\n\r]/) !== null) {
            var result8 = input.charAt(pos);
            pos++;
        } else {
            var result8 = null;
            if (reportMatchFailures) {
                matchFailed("[^'\\\\\\n\\r]");
            }
        }
        if (result8 !== null) {
            var result6 = result8;
        } else {
            var result7 = parse_ECHAR();
            if (result7 !== null) {
                var result6 = result7;
            } else {
                var result6 = null;;
            };
        }
        while (result6 !== null) {
            result4.push(result6);
            if (input.substr(pos).match(/^[^'\\\n\r]/) !== null) {
                var result8 = input.charAt(pos);
                pos++;
            } else {
                var result8 = null;
                if (reportMatchFailures) {
                    matchFailed("[^'\\\\\\n\\r]");
                }
            }
            if (result8 !== null) {
                var result6 = result8;
            } else {
                var result7 = parse_ECHAR();
                if (result7 !== null) {
                    var result6 = result7;
                } else {
                    var result6 = null;;
                };
            }
        }
        if (result4 !== null) {
            if (input.substr(pos, 1) === "'") {
                var result5 = "'";
                pos += 1;
            } else {
                var result5 = null;
                if (reportMatchFailures) {
                    matchFailed("\"'\"");
                }
            }
            if (result5 !== null) {
                var result1 = [result3, result4, result5];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(content) { return flattenString(content) })(result1[1])
```

```
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[139] STRING_LITERAL1");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_STRING_LITERAL2() {
        var cacheKey = 'STRING_LITERAL2@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 1) === "\"") {
            var result3 = "\"";
            pos += 1;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"\\\"\"");
            }
        }
        if (result3 !== null) {
            var result4 = [];
            if (input.substr(pos).match(/^[^"\\\n\r]/) !== null) {
                var result8 = input.charAt(pos);
                pos++;
            } else {
                var result8 = null;
                if (reportMatchFailures) {
                    matchFailed("[^\"\\\\\\n\\r]");
                }
            }
            if (result8 !== null) {
                var result6 = result8;
            } else {
                var result7 = parse_ECHAR();
                if (result7 !== null) {
                    var result6 = result7;
                } else {
                    var result6 = null;;
                };
            }
            while (result6 !== null) {
                result4.push(result6);
                if (input.substr(pos).match(/^[^"\\\n\r]/) !== null) {
                    var result8 = input.charAt(pos);
                    pos++;
                } else {
```

```
                    var result8 = null;
                    if (reportMatchFailures) {
                        matchFailed("[^\"\\\\\\n\\r]");
                    }
                }
                if (result8 !== null) {
                    var result6 = result8;
                } else {
                    var result7 = parse_ECHAR();
                    if (result7 !== null) {
                        var result6 = result7;
                    } else {
                        var result6 = null;;
                    };
                }
            }
            if (result4 !== null) {
                if (input.substr(pos, 1) === "\"") {
                    var result5 = "\"";
                    pos += 1;
                } else {
                    var result5 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"\\\"\"");
                    }
                }
                if (result5 !== null) {
                    var result1 = [result3, result4, result5];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(content) { return flattenString(content) })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[140] STRING_LITERAL2");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_STRING_LITERAL_LONG1() {
        var cacheKey = 'STRING_LITERAL_LONG1@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
```

```
}

var savedReportMatchFailures = reportMatchFailures;
reportMatchFailures = false;
var savedPos0 = pos;
var savedPos1 = pos;
if (input.substr(pos, 3) === "'''") {
    var result3 = "'''";
    pos += 3;
} else {
    var result3 = null;
    if (reportMatchFailures) {
        matchFailed("\"'''\"");
    }
}
if (result3 !== null) {
    var result4 = [];
    if (input.substr(pos).match(/^[^'\\]/) !== null) {
        var result8 = input.charAt(pos);
        pos++;
    } else {
        var result8 = null;
        if (reportMatchFailures) {
            matchFailed("[^'\\\\]");
        }
    }
    if (result8 !== null) {
        var result6 = result8;
    } else {
        var result7 = parse_ECHAR();
        if (result7 !== null) {
            var result6 = result7;
        } else {
            var result6 = null;;
        };
    }
    while (result6 !== null) {
        result4.push(result6);
        if (input.substr(pos).match(/^[^'\\]/) !== null) {
            var result8 = input.charAt(pos);
            pos++;
        } else {
            var result8 = null;
            if (reportMatchFailures) {
                matchFailed("[^'\\\\]");
            }
        }
        if (result8 !== null) {
            var result6 = result8;
        } else {
            var result7 = parse_ECHAR();
            if (result7 !== null) {
                var result6 = result7;
            } else {
                var result6 = null;;
            };
        }
    }
    if (result4 !== null) {
        if (input.substr(pos, 3) === "'''") {
            var result5 = "'''";
            pos += 3;
        } else {
            var result5 = null;
            if (reportMatchFailures) {
                matchFailed("\"'''\"");
            }
```

```
                }
                if (result5 !== null) {
                    var result1 = [result3, result4, result5];
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(content) { return flattenString(content) })(result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[141] STRING_LITERAL_LONG1");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_STRING_LITERAL_LONG2() {
        var cacheKey = 'STRING_LITERAL_LONG2@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        if (input.substr(pos, 3) === "\"\"\"") {
            var result3 = "\"\"\"";
            pos += 3;
        } else {
            var result3 = null;
            if (reportMatchFailures) {
                matchFailed("\"\\\"\\\"\\\"\"");
            }
        }
        if (result3 !== null) {
            var result4 = [];
            if (input.substr(pos).match(/^[^"\\]/) !== null) {
                var result8 = input.charAt(pos);
                pos++;
            } else {
                var result8 = null;
                if (reportMatchFailures) {
                    matchFailed("[^\"\\\\]");
                }
```

```
        }
        if (result8 !== null) {
            var result6 = result8;
        } else {
            var result7 = parse_ECHAR();
            if (result7 !== null) {
                var result6 = result7;
            } else {
                var result6 = null;;
            };
        }
        while (result6 !== null) {
            result4.push(result6);
            if (input.substr(pos).match(/^[^"\\]/) !== null) {
                var result8 = input.charAt(pos);
                pos++;
            } else {
                var result8 = null;
                if (reportMatchFailures) {
                    matchFailed("[^\"\\\\]");
                }
            }
            if (result8 !== null) {
                var result6 = result8;
            } else {
                var result7 = parse_ECHAR();
                if (result7 !== null) {
                    var result6 = result7;
                } else {
                    var result6 = null;;
                };
            }
        }
        if (result4 !== null) {
            if (input.substr(pos, 3) === "\"\"\"") {
                var result5 = "\"\"\"";
                pos += 3;
            } else {
                var result5 = null;
                if (reportMatchFailures) {
                    matchFailed("\"\\\"\\\"\\\"\"");
                }
            }
            if (result5 !== null) {
                var result1 = [result3, result4, result5];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
    } else {
        var result1 = null;
        pos = savedPos1;
    }
    var result2 = result1 !== null
        ? (function(content) { return flattenString(content) })(result1[1])
        : null;
    if (result2 !== null) {
        var result0 = result2;
    } else {
        var result0 = null;
        pos = savedPos0;
    }
    reportMatchFailures = savedReportMatchFailures;
```

```
        if (reportMatchFailures && result0 === null) {
            matchFailed("[142] STRING_LITERAL_LONG2");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ECHAR() {
        var cacheKey = 'ECHAR@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        if (input.substr(pos, 1) === "\\") {
            var result1 = "\\";
            pos += 1;
        } else {
            var result1 = null;
            if (reportMatchFailures) {
                matchFailed("\"\\\\\"");
            }
        }
        if (result1 !== null) {
            if (input.substr(pos).match(/^[tbnrf"']/) !== null) {
                var result2 = input.charAt(pos);
                pos++;
            } else {
                var result2 = null;
                if (reportMatchFailures) {
                    matchFailed("[tbnrf\"']");
                }
            }
            if (result2 !== null) {
                var result0 = [result1, result2];
            } else {
                var result0 = null;
                pos = savedPos0;
            }
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[143] ECHAR");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_NIL() {
        var cacheKey = 'NIL@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
```

```
                    pos = cachedResult.nextPos;
                    return cachedResult.result;
                }

                var savedReportMatchFailures = reportMatchFailures;
                reportMatchFailures = false;
                var savedPos0 = pos;
                var savedPos1 = pos;
                if (input.substr(pos, 1) === "(") {
                    var result3 = "(";
                    pos += 1;
                } else {
                    var result3 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"(\"");
                    }
                }
                if (result3 !== null) {
                    var result4 = [];
                    var result6 = parse_WS();
                    while (result6 !== null) {
                        result4.push(result6);
                        var result6 = parse_WS();
                    }
                    if (result4 !== null) {
                        if (input.substr(pos, 1) === ")") {
                            var result5 = ")";
                            pos += 1;
                        } else {
                            var result5 = null;
                            if (reportMatchFailures) {
                                matchFailed("\")\"");
                            }
                        }
                        if (result5 !== null) {
                            var result1 = [result3, result4, result5];
                        } else {
                            var result1 = null;
                            pos = savedPos1;
                        }
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function() {

                    return  {token: "triplesnodecollection",
                        triplesContext:[],
                        chainSubject:[{token:'uri', value:"http://www.w3.org/1999/02/22-rdf-syntax-
ns#nil"}]};
                })()
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[144] NIL");
                }
```

```
        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_WS() {
        var cacheKey = 'WS@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        if (input.substr(pos).match(/^[ ]/) !== null) {
            var result5 = input.charAt(pos);
            pos++;
        } else {
            var result5 = null;
            if (reportMatchFailures) {
                matchFailed("[ ]");
            }
        }
        if (result5 !== null) {
            var result0 = result5;
        } else {
            if (input.substr(pos).match(/^[ ]/) !== null) {
                var result4 = input.charAt(pos);
                pos++;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("[  ]");
                }
            }
            if (result4 !== null) {
                var result0 = result4;
            } else {
                if (input.substr(pos).match(/^[\r]/) !== null) {
                    var result3 = input.charAt(pos);
                    pos++;
                } else {
                    var result3 = null;
                    if (reportMatchFailures) {
                        matchFailed("[\\r]");
                    }
                }
                if (result3 !== null) {
                    var result0 = result3;
                } else {
                    if (input.substr(pos).match(/^[\n]/) !== null) {
                        var result2 = input.charAt(pos);
                        pos++;
                    } else {
                        var result2 = null;
                        if (reportMatchFailures) {
                            matchFailed("[\\n]");
                        }
                    }
                    if (result2 !== null) {
                        var result0 = result2;
                    } else {
                        var result1 = parse_COMMENT();
```

```
                    if (result1 !== null) {
                        var result0 = result1;
                    } else {
                        var result0 = null;;
                    };
                };
            };
        };
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[145] WS");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_COMMENT() {
        var cacheKey = 'COMMENT@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        if (input.substr(pos, 1) === "#") {
            var result1 = "#";
            pos += 1;
        } else {
            var result1 = null;
            if (reportMatchFailures) {
                matchFailed("\"#\"");
            }
        }
        if (result1 !== null) {
            var result2 = [];
            if (input.substr(pos).match(/^[^\n\r]/) !== null) {
                var result3 = input.charAt(pos);
                pos++;
            } else {
                var result3 = null;
                if (reportMatchFailures) {
                    matchFailed("[^\\n\\r]");
                }
            }
            while (result3 !== null) {
                result2.push(result3);
                if (input.substr(pos).match(/^[^\n\r]/) !== null) {
                    var result3 = input.charAt(pos);
                    pos++;
                } else {
                    var result3 = null;
                    if (reportMatchFailures) {
                        matchFailed("[^\\n\\r]");
                    }
                }
            }
            if (result2 !== null) {
                var result0 = [result1, result2];
            } else {
```

```
                var result0 = null;
                pos = savedPos0;
            }
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed(" COMMENT");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_ANON() {
        var cacheKey = 'ANON@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        if (input.substr(pos, 1) === "[") {
            var result1 = "[";
            pos += 1;
        } else {
            var result1 = null;
            if (reportMatchFailures) {
                matchFailed("\"[\"");
            }
        }
        if (result1 !== null) {
            var result2 = [];
            var result4 = parse_WS();
            while (result4 !== null) {
                result2.push(result4);
                var result4 = parse_WS();
            }
            if (result2 !== null) {
                if (input.substr(pos, 1) === "]") {
                    var result3 = "]";
                    pos += 1;
                } else {
                    var result3 = null;
                    if (reportMatchFailures) {
                        matchFailed("\"]\"");
                    }
                }
                if (result3 !== null) {
                    var result0 = [result1, result2, result3];
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
            } else {
                var result0 = null;
                pos = savedPos0;
            }
        } else {
```

```
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[146] ANON");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PN_CHARS_BASE() {
        var cacheKey = 'PN_CHARS_BASE@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        if (input.substr(pos).match(/^[A-Z]/) !== null) {
            var result14 = input.charAt(pos);
            pos++;
        } else {
            var result14 = null;
            if (reportMatchFailures) {
                matchFailed("[A-Z]");
            }
        }
        if (result14 !== null) {
            var result0 = result14;
        } else {
            if (input.substr(pos).match(/^[a-z]/) !== null) {
                var result13 = input.charAt(pos);
                pos++;
            } else {
                var result13 = null;
                if (reportMatchFailures) {
                    matchFailed("[a-z]");
                }
            }
            if (result13 !== null) {
                var result0 = result13;
            } else {
                if (input.substr(pos).match(/^[\xC0-\xD6]/) !== null) {
                    var result12 = input.charAt(pos);
                    pos++;
                } else {
                    var result12 = null;
                    if (reportMatchFailures) {
                        matchFailed("[\\xC0-\\xD6]");
                    }
                }
                if (result12 !== null) {
                    var result0 = result12;
                } else {
                    if (input.substr(pos).match(/^[\xD8-\xF6]/) !== null) {
                        var result11 = input.charAt(pos);
                        pos++;
                    } else {
                        var result11 = null;
                        if (reportMatchFailures) {
```

```
                                        matchFailed("[\\xD8-\\xF6]");
                                    }
                                }
                                if (result11 !== null) {
                                    var result0 = result11;
                                } else {
                                    if (input.substr(pos).match(/^[\xF8-\u02FF]/) !== null) {
                                        var result10 = input.charAt(pos);
                                        pos++;
                                    } else {
                                        var result10 = null;
                                        if (reportMatchFailures) {
                                            matchFailed("[\\xF8-\\u02FF]");
                                        }
                                    }
                                    if (result10 !== null) {
                                        var result0 = result10;
                                    } else {
                                        if (input.substr(pos).match(/^[\u0370-\u037D]/) !== null) {
                                            var result9 = input.charAt(pos);
                                            pos++;
                                        } else {
                                            var result9 = null;
                                            if (reportMatchFailures) {
                                                matchFailed("[\\u0370-\\u037D]");
                                            }
                                        }
                                        if (result9 !== null) {
                                            var result0 = result9;
                                        } else {
                                            if (input.substr(pos).match(/^[\u037F-\u1FFF]/) !== null) {
                                                var result8 = input.charAt(pos);
                                                pos++;
                                            } else {
                                                var result8 = null;
                                                if (reportMatchFailures) {
                                                    matchFailed("[\\u037F-\\u1FFF]");
                                                }
                                            }
                                            if (result8 !== null) {
                                                var result0 = result8;
                                            } else {
                                                if (input.substr(pos).match(/^[\u200C-\u200D]/) !== null)
{
                                                    var result7 = input.charAt(pos);
                                                    pos++;
                                                } else {
                                                    var result7 = null;
                                                    if (reportMatchFailures) {
                                                        matchFailed("[\\u200C-\\u200D]");
                                                    }
                                                }
                                                if (result7 !== null) {
                                                    var result0 = result7;
                                                } else {
                                                    if (input.substr(pos).match(/^[\u2070-\u218F]/) !==
null) {
                                                        var result6 = input.charAt(pos);
                                                        pos++;
                                                    } else {
                                                        var result6 = null;
                                                        if (reportMatchFailures) {
                                                            matchFailed("[\\u2070-\\u218F]");
                                                        }
                                                    }
                                                    if (result6 !== null) {
                                                        var result0 = result6;
```

```
== null) {
                                                                    } else {
                                                                        if (input.substr(pos).match(/^[\u2C00-\u2FEF]/) !
                                                                            var result5 = input.charAt(pos);
                                                                            pos++;
                                                                        } else {
                                                                            var result5 = null;
                                                                            if (reportMatchFailures) {
                                                                                matchFailed("[\\u2C00-\\u2FEF]");
                                                                            }
                                                                        }
                                                                        if (result5 !== null) {
                                                                            var result0 = result5;
                                                                        } else {
                                                                            if (input.substr(pos).match(/^[\u3001-\uD7FF]/
) !== null) {
                                                                                var result4 = input.charAt(pos);
                                                                                pos++;
                                                                            } else {
                                                                                var result4 = null;
                                                                                if (reportMatchFailures) {
                                                                                    matchFailed("[\\u3001-\\uD7FF]");
                                                                                }
                                                                            }
                                                                            if (result4 !== null) {
                                                                                var result0 = result4;
                                                                            } else {
                                                                                if (input.substr(pos).match(/^[\uF900-
\uFDCF]/) !== null) {
                                                                                    var result3 = input.charAt(pos);
                                                                                    pos++;
                                                                                } else {
                                                                                    var result3 = null;
                                                                                    if (reportMatchFailures) {
                                                                                        matchFailed("[\\uF900-\\uFDCF]");
                                                                                    }
                                                                                }
                                                                                if (result3 !== null) {
                                                                                    var result0 = result3;
                                                                                } else {
                                                                                    if (input.substr(pos).match(/^[\uFDF0-
\uFFFD]/) !== null) {
                                                                                        var result2 = input.charAt(pos);
                                                                                        pos++;
                                                                                    } else {
                                                                                        var result2 = null;
                                                                                        if (reportMatchFailures) {
                                                                                            matchFailed("[\\uFDF0-\
\uFFFD]");
                                                                                        }
                                                                                    }
                                                                                    if (result2 !== null) {
                                                                                        var result0 = result2;
                                                                                    } else {
                                                                                        if (input.substr(pos).match(/^
[\u1000-\uEFFF]/) !== null) {
                                                                                            var result1 = input.charAt
(pos);
                                                                                            pos++;
                                                                                        } else {
                                                                                            var result1 = null;
                                                                                            if (reportMatchFailures) {
                                                                                                matchFailed("[\\u1000-\
\uEFFF]");
                                                                                            }
                                                                                        }
                                                                                        if (result1 !== null) {
```

```
                                                                    var result0 = result1;
                                                                } else {
                                                                    var result0 = null;;
                                                                };
                                                            };
                                                        };
                                                    };
                                                };
                                            };
                                        };
                                    };
                                };
                            };
                        };
                    }
                    reportMatchFailures = savedReportMatchFailures;
                    if (reportMatchFailures && result0 === null) {
                        matchFailed("[147] PN_CHARS_BASE");
                    }

                    cache[cacheKey] = {
                        nextPos: pos,
                        result:  result0
                    };
                    return result0;
                }

                function parse_PN_CHARS_U() {
                    var cacheKey = 'PN_CHARS_U@' + pos;
                    var cachedResult = cache[cacheKey];
                    if (cachedResult) {
                        pos = cachedResult.nextPos;
                        return cachedResult.result;
                    }

                    var savedReportMatchFailures = reportMatchFailures;
                    reportMatchFailures = false;
                    var result2 = parse_PN_CHARS_BASE();
                    if (result2 !== null) {
                        var result0 = result2;
                    } else {
                        if (input.substr(pos, 1) === "_") {
                            var result1 = "_";
                            pos += 1;
                        } else {
                            var result1 = null;
                            if (reportMatchFailures) {
                                matchFailed("\"_\"");
                            }
                        }
                        if (result1 !== null) {
                            var result0 = result1;
                        } else {
                            var result0 = null;;
                        };
                    }
                    reportMatchFailures = savedReportMatchFailures;
                    if (reportMatchFailures && result0 === null) {
                        matchFailed("[148] PN_CHARS_U");
                    }

                    cache[cacheKey] = {
                        nextPos: pos,
                        result:  result0
                    };
```

```
        return result0;
    }

    function parse_VARNAME() {
        var cacheKey = 'VARNAME@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result12 = parse_PN_CHARS_U();
        if (result12 !== null) {
            var result3 = result12;
        } else {
            if (input.substr(pos).match(/^[0-9]/) !== null) {
                var result11 = input.charAt(pos);
                pos++;
            } else {
                var result11 = null;
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
            if (result11 !== null) {
                var result3 = result11;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
            var result4 = [];
            var result10 = parse_PN_CHARS_U();
            if (result10 !== null) {
                var result5 = result10;
            } else {
                if (input.substr(pos).match(/^[0-9]/) !== null) {
                    var result9 = input.charAt(pos);
                    pos++;
                } else {
                    var result9 = null;
                    if (reportMatchFailures) {
                        matchFailed("[0-9]");
                    }
                }
                if (result9 !== null) {
                    var result5 = result9;
                } else {
                    if (input.substr(pos).match(/^[\xB7]/) !== null) {
                        var result8 = input.charAt(pos);
                        pos++;
                    } else {
                        var result8 = null;
                        if (reportMatchFailures) {
                            matchFailed("[\\xB7]");
                        }
                    }
                    if (result8 !== null) {
                        var result5 = result8;
                    } else {
                        if (input.substr(pos).match(/^[\u0300-\u036F]/) !== null) {
                            var result7 = input.charAt(pos);
                            pos++;
```

```
                    } else {
                        var result7 = null;
                        if (reportMatchFailures) {
                            matchFailed("[\\u0300-\\u036F]");
                        }
                    }
                    if (result7 !== null) {
                        var result5 = result7;
                    } else {
                        if (input.substr(pos).match(/^[\u203F-\u2040]/) !== null) {
                            var result6 = input.charAt(pos);
                            pos++;
                        } else {
                            var result6 = null;
                            if (reportMatchFailures) {
                                matchFailed("[\\u203F-\\u2040]");
                            }
                        }
                        if (result6 !== null) {
                            var result5 = result6;
                        } else {
                            var result5 = null;;
                        };
                    };
                };
            };
        }
        while (result5 !== null) {
            result4.push(result5);
            var result10 = parse_PN_CHARS_U();
            if (result10 !== null) {
                var result5 = result10;
            } else {
                if (input.substr(pos).match(/^[0-9]/) !== null) {
                    var result9 = input.charAt(pos);
                    pos++;
                } else {
                    var result9 = null;
                    if (reportMatchFailures) {
                        matchFailed("[0-9]");
                    }
                }
                if (result9 !== null) {
                    var result5 = result9;
                } else {
                    if (input.substr(pos).match(/^[\xB7]/) !== null) {
                        var result8 = input.charAt(pos);
                        pos++;
                    } else {
                        var result8 = null;
                        if (reportMatchFailures) {
                            matchFailed("[\\xB7]");
                        }
                    }
                    if (result8 !== null) {
                        var result5 = result8;
                    } else {
                        if (input.substr(pos).match(/^[\u0300-\u036F]/) !== null) {
                            var result7 = input.charAt(pos);
                            pos++;
                        } else {
                            var result7 = null;
                            if (reportMatchFailures) {
                                matchFailed("[\\u0300-\\u036F]");
                            }
                        }
                        if (result7 !== null) {
```

```
                                var result5 = result7;
                            } else {
                                if (input.substr(pos).match(/^[\u203F-\u2040]/) !== null) {
                                    var result6 = input.charAt(pos);
                                    pos++;
                                } else {
                                    var result6 = null;
                                    if (reportMatchFailures) {
                                        matchFailed("[\\u203F-\\u2040]");
                                    }
                                }
                                if (result6 !== null) {
                                    var result5 = result6;
                                } else {
                                    var result5 = null;;
                                };
                            };
                        };
                    };
                }
            }
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(init, rpart) { return init+rpart.join('') })(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[149] VARNAME");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PN_CHARS() {
        var cacheKey = 'PN_CHARS@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var result6 = parse_PN_CHARS_U();
        if (result6 !== null) {
            var result0 = result6;
        } else {
            if (input.substr(pos, 1) === "-") {
```

```
            var result5 = "-";
            pos += 1;
        } else {
            var result5 = null;
            if (reportMatchFailures) {
                matchFailed("\"-\"");
            }
        }
        if (result5 !== null) {
            var result0 = result5;
        } else {
            if (input.substr(pos).match(/^[0-9]/) !== null) {
                var result4 = input.charAt(pos);
                pos++;
            } else {
                var result4 = null;
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
            if (result4 !== null) {
                var result0 = result4;
            } else {
                if (input.substr(pos).match(/^[\xB7]/) !== null) {
                    var result3 = input.charAt(pos);
                    pos++;
                } else {
                    var result3 = null;
                    if (reportMatchFailures) {
                        matchFailed("[\\xB7]");
                    }
                }
                if (result3 !== null) {
                    var result0 = result3;
                } else {
                    if (input.substr(pos).match(/^[\u0300-\u036F]/) !== null) {
                        var result2 = input.charAt(pos);
                        pos++;
                    } else {
                        var result2 = null;
                        if (reportMatchFailures) {
                            matchFailed("[\\u0300-\\u036F]");
                        }
                    }
                    if (result2 !== null) {
                        var result0 = result2;
                    } else {
                        if (input.substr(pos).match(/^[\u203F-\u2040]/) !== null) {
                            var result1 = input.charAt(pos);
                            pos++;
                        } else {
                            var result1 = null;
                            if (reportMatchFailures) {
                                matchFailed("[\\u203F-\\u2040]");
                            }
                        }
                        if (result1 !== null) {
                            var result0 = result1;
                        } else {
                            var result0 = null;;
                        };
                    };
                };
            };
        };
    }
    reportMatchFailures = savedReportMatchFailures;
```

```
        if (reportMatchFailures && result0 === null) {
            matchFailed("[150] PN_CHARS");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PN_PREFIX() {
        var cacheKey = 'PN_PREFIX@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result3 = parse_PN_CHARS_BASE();
        if (result3 !== null) {
            var result4 = [];
            var result7 = parse_PN_CHARS();
            if (result7 !== null) {
                var result5 = result7;
            } else {
                if (input.substr(pos, 1) === ".") {
                    var result6 = ".";
                    pos += 1;
                } else {
                    var result6 = null;
                    if (reportMatchFailures) {
                        matchFailed("\".\"");
                    }
                }
                if (result6 !== null) {
                    var result5 = result6;
                } else {
                    var result5 = null;;
                };
            }
            while (result5 !== null) {
                result4.push(result5);
                var result7 = parse_PN_CHARS();
                if (result7 !== null) {
                    var result5 = result7;
                } else {
                    if (input.substr(pos, 1) === ".") {
                        var result6 = ".";
                        pos += 1;
                    } else {
                        var result6 = null;
                        if (reportMatchFailures) {
                            matchFailed("\".\"");
                        }
                    }
                    if (result6 !== null) {
                        var result5 = result6;
                    } else {
                        var result5 = null;;
                    };
                }
            }
```

```
            if (result4 !== null) {
                var result1 = [result3, result4];
            } else {
                var result1 = null;
                pos = savedPos1;
            }
        } else {
            var result1 = null;
            pos = savedPos1;
        }
        var result2 = result1 !== null
            ? (function(base, rest) { if(rest[rest.length-1] == '.'){
            throw new Error("Wrong PN_PREFIX, cannot finish with '.'")
        } else {
            return base + rest.join('');
        }})(result1[0], result1[1])
            : null;
        if (result2 !== null) {
            var result0 = result2;
        } else {
            var result0 = null;
            pos = savedPos0;
        }
        reportMatchFailures = savedReportMatchFailures;
        if (reportMatchFailures && result0 === null) {
            matchFailed("[151] PN_PREFIX");
        }

        cache[cacheKey] = {
            nextPos: pos,
            result:  result0
        };
        return result0;
    }

    function parse_PN_LOCAL() {
        var cacheKey = 'PN_LOCAL@' + pos;
        var cachedResult = cache[cacheKey];
        if (cachedResult) {
            pos = cachedResult.nextPos;
            return cachedResult.result;
        }

        var savedReportMatchFailures = reportMatchFailures;
        reportMatchFailures = false;
        var savedPos0 = pos;
        var savedPos1 = pos;
        var result7 = parse_PN_CHARS_U();
        if (result7 !== null) {
            var result3 = result7;
        } else {
            if (input.substr(pos).match(/^[0-9]/) !== null) {
                var result6 = input.charAt(pos);
                pos++;
            } else {
                var result6 = null;
                if (reportMatchFailures) {
                    matchFailed("[0-9]");
                }
            }
            if (result6 !== null) {
                var result3 = result6;
            } else {
                var result3 = null;;
            };
        }
        if (result3 !== null) {
```

```
                    var result4 = [];
                    var result5 = parse_PN_CHARS();
                    while (result5 !== null) {
                        result4.push(result5);
                        var result5 = parse_PN_CHARS();
                    }
                    if (result4 !== null) {
                        var result1 = [result3, result4];
                    } else {
                        var result1 = null;
                        pos = savedPos1;
                    }
                } else {
                    var result1 = null;
                    pos = savedPos1;
                }
                var result2 = result1 !== null
                    ? (function(base, rest) {
                    return base + rest.join('');
                })(result1[0], result1[1])
                    : null;
                if (result2 !== null) {
                    var result0 = result2;
                } else {
                    var result0 = null;
                    pos = savedPos0;
                }
                reportMatchFailures = savedReportMatchFailures;
                if (reportMatchFailures && result0 === null) {
                    matchFailed("[152] PN_LOCAL");
                }

                cache[cacheKey] = {
                    nextPos: pos,
                    result:  result0
                };
                return result0;
            }

            function buildErrorMessage() {
                function buildExpected(failuresExpected) {
                    failuresExpected.sort();

                    var lastFailure = null;
                    var failuresExpectedUnique = [];
                    for (var i = 0; i < failuresExpected.length; i++) {
                        if (failuresExpected[i] !== lastFailure) {
                            failuresExpectedUnique.push(failuresExpected[i]);
                            lastFailure = failuresExpected[i];
                        }
                    }

                    switch (failuresExpectedUnique.length) {
                        case 0:
                            return 'end of input';
                        case 1:
                            return failuresExpectedUnique[0];
                        default:
                            return failuresExpectedUnique.slice(0, failuresExpectedUnique.length -
1).join(', ')
                                + ' or '
                                + failuresExpectedUnique[failuresExpectedUnique.length - 1];
                    }
                }

                var expected = buildExpected(rightmostMatchFailuresExpected);
                var actualPos = Math.max(pos, rightmostMatchFailuresPos);
```

```
    var actual = actualPos < input.length
        ? quote(input.charAt(actualPos))
        : 'end of input';

    return 'Expected ' + expected + ' but ' + actual + ' found.';
}

function computeErrorPosition() {
    /*
     * The first idea was to use |String.split| to break the input up to the
     * error position along newlines and derive the line and column from
     * there. However IE's |split| implementation is so broken that it was
     * enough to prevent it.
     */

    var line = 1;
    var column = 1;
    var seenCR = false;

    for (var i = 0; i <  rightmostMatchFailuresPos; i++) {
        var ch = input.charAt(i);
        if (ch === '\n') {
            if (!seenCR) { line++; }
            column = 1;
            seenCR = false;
        } else if (ch === '\r' | ch === '\u2028' || ch === '\u2029') {
            line++;
            column = 1;
            seenCR = true;
        } else {
            column++;
            seenCR = false;
        }
    }

    return { line: line, column: column };
}



var flattenString = function(arrs) {

    var acum ="";

    for(var i=0; i< arrs.length; i++) {

        if(typeof(arrs[i])==='string') {

            acum = acum + arrs[i];

        } else {

            acum = acum + arrs[i].join('');

        }

    }



    return acum;

}
```

```
var GlobalBlankNodeCounter = 0;


var prefixes = {};


var registerPrefix = function(prefix, uri) {
    prefixes[prefix] = uri;
}


var registerDefaultPrefix = function(uri) {
    prefixes[null] = uri;
}


var arrayToString = function(array) {
    var tmp = "";
    for(var i=0; i<array.length; i++) {
        tmp = tmp + array[i];
    }


    return tmp.toUpperCase();
}


var result = parseFunctions[startRule]();

/*
 * The parser is now in one of the following three states:
 *
 * 1. The parser successfully parsed the whole input.
 *
 *    - |result !== null|
 *    - |pos === input.length|
 *    - |rightmostMatchFailuresExpected| may or may not contain something
 *
 * 2. The parser successfully parsed only a part of the input.
 *
 *    - |result !== null|
 *    - |pos < input.length|
 *    - |rightmostMatchFailuresExpected| may or may not contain something
 *
 * 3. The parser did not successfully parse any part of the input.
 *
 *   - |result === null|
 *   - |pos === 0|
 *   - |rightmostMatchFailuresExpected| contains at least one failure
 *
 * All code following this comment (including called functions) must
```

```
                 * handle these states.
                 */
                if (result === null || pos !== input.length) {
                    var errorPosition = computeErrorPosition();
                    var errorMsg = buildErrorMessage();
                    //console.log("ERROR IN SPARQL PARSER", errorMsg, errorPosition);
                    throw new this.SyntaxError(
                        errorMsg,
                        errorPosition.line,
                        errorPosition.column
                    );
                }

                return result;
            },

            /* Returns the parser source code. */
            toSource: function() { return this._source; }
        };

        /* Thrown when a parser encounters a syntax error. */

        result.SyntaxError = function(message, line, column) {
            this.name = 'SyntaxError';
            this.message = message;
            this.line = line;
            this.column = column;
        };

        result.SyntaxError.prototype = Error.prototype;

        return result;
    })();
    return SparqlParser;
})define([], function () {
    var InMemoryBTree =  {},
        left = -1,
        right = 1;
    /**
     * @doc
     * Implementation based on <http://www.gossamer-threads.com/lists/linux/kernel/667935>
     *
     */

    /**
     * Tree
     *
     * Implements the interface of BinarySearchTree.Tree
     *
     * An implementation of an in memory B-Tree.
     */

    InMemoryBTree.Tree = function(order) {
        if(arguments.length != 0) {
            this.order = order;
            this.root = this._allocateNode();
            this.root.isLeaf = true;
            this.root.level = 0;
            this._diskWrite(this.root);
            this._updateRootNode(this.root);

            this.comparator = function(a,b) {
                if(a < b) {
                    return -1;
                } else if(a > b){
                    return 1;
                } else {
```

```
                return 0;
            }
        };
        this.merger = null;
    }
};

/**
 * Creates the new node.
 *
 * This class can be overwritten by different versions of
 * the tree t select the right kind of node to be used
 *
 * @returns the new alloacted node
 */
InMemoryBTree.Tree.prototype._allocateNode = function () {
    return new InMemoryBTree.Node();
};

/**
 * _diskWrite
 *
 * Persists the node to secondary memory.
 */
InMemoryBTree.Tree.prototype._diskWrite= function(node) {
    // dummy implementation;
    // no-op
};


/**
 * _diskRead
 *
 * Retrieves a node from secondary memory using the provided
 * pointer
 */
InMemoryBTree.Tree.prototype._diskRead = function(pointer) {
    // dummy implementation;
    // no-op
    return pointer;
};


InMemoryBTree.Tree.prototype._diskDelete= function(node) {
    // dummy implmentation
    // no-op
};

/**
 * _updateRootNode
 *
 * Updates the pointer to the root node stored in disk.
 */
InMemoryBTree.Tree.prototype._updateRootNode = function(node) {
    // dummy implementation;
    // no-op
    return node;
};

InMemoryBTree.Tree.prototype.clear = function() {
    this.root = this._allocateNode();
    this.root.isLeaf = true;
    this.root.level = 0;
    this._updateRootNode(this.root);
};

/**
```

```
 * search
 *
 * Retrieves the node matching the given value.
 * If no node is found, null is returned.
 */
InMemoryBTree.Tree.prototype.search = function(key, checkExists) {
    var searching = true;
    var node = this.root;

    while(searching) {
        var idx = 0;
        while(idx < node.numberActives && this.comparator(key, node.keys[idx].key) === 1) {
            idx++;
        }

        if(idx < node.numberActives && this.comparator(node.keys[idx].key,key) === 0) {
            if(checkExists != null && checkExists == true) {
                return true;
            } else {
                return node.keys[idx].data;
            }
        } else {
            if(node.isLeaf === true) {
                searching = false;
            } else {
                node = this._diskRead(node.children[idx]);
            }
        }
    }

    return null;
};


/**
 * walk
 * Applies a function to all the nodes key and data in the the
 * tree in key order.
 */
InMemoryBTree.Tree.prototype.walk = function(f) {
    this._walk(f,this.root);
};

InMemoryBTree.Tree.prototype._walk = function(f,node) {
    if(node.isLeaf) {
        for(var i=0; i<node.numberActives; i++) {
            f(node.keys[i]);
        }
    } else {
        for(var i=0; i<node.numberActives; i++) {
            this._walk(f,this._diskRead(node.children[i]));
            f(node.keys[i]);
        }
        this._walk(f,this._diskRead(node.children[node.numberActives]));
    }
};

/**
 * walkNodes
 * Applies a function to all the nodes in the the
 * tree in key order.
 */
InMemoryBTree.Tree.prototype.walkNodes = function(f) {
    this._walkNodes(f,this.root);
};

InMemoryBTree.Tree.prototype._walkNodes = function(f,node) {
```

```
        if(node.isLeaf) {
            f(node);
        } else {
            f(node);
            for(var i=0; i<node.numberActives; i++) {
                this._walkNodes(f,this._diskRead(node.children[i]));
            }
            this._walkNodes(f,this._diskRead(node.children[node.numberActives]));
        }
    };

    /**
     * _splitChild
     *
     * Split the child node and adjusts the parent.
     */
    InMemoryBTree.Tree.prototype._splitChild = function(parent, index, child) {
        var newChild = this._allocateNode();
        newChild.isLeaf = child.isLeaf;
        newChild.level = child.level;
        newChild.numberActives = this.order - 1;

        // Copy the higher order keys to the new child
        var newParentChild = child.keys[this.order-1];
        child.keys[this.order-1] = null;

        for(var i=0; i< this.order-1; i++) {
            newChild.keys[i]=child.keys[i+this.order];
            child.keys[i+this.order] = null;
            if(!child.isLeaf) {
                newChild.children[i] = child.children[i+this.order];
                child.children[i+this.order] = null;
            }
        }

        // Copy the last child pointer
        if(!child.isLeaf) {
            newChild.children[i] = child.children[i+this.order];
            child.children[i+this.order] = null;
        }

        child.numberActives = this.order - 1;


        for(i = parent.numberActives + 1; i>index+1; i--) {
            parent.children[i] = parent.children[i-1];
        }

        parent.children[index+1] = newChild;

        for(i = parent.numberActives; i>index; i--) {
            parent.keys[i] = parent.keys[i-1];
        }

        parent.keys[index] = newParentChild;
        parent.numberActives++;

        this._diskWrite(newChild);
        this._diskWrite(parent);
        this._diskWrite(child);
    };

    /**
     * insert
     *
     * Creates a new node with value key and data and inserts it
     * into the tree.
```

```
 */
InMemoryBTree.Tree.prototype.insert = function(key,data) {
    if(this.root.numberActives === (2 * this.order - 1)) {
        var newRoot = this._allocateNode();
        newRoot.isLeaf = false;
        newRoot.level = this.root.level + 1;
        newRoot.numberActives = 0;
        newRoot.children[0] = this.root;

        this._splitChild(newRoot, 0, this.root);
        this.root = newRoot;
        this._updateRootNode(this.root);
        this._insertNonFull(newRoot, key, data);
    } else {
        this._insertNonFull(this.root, key, data);
    }
};

/**
 * _insertNonFull
 *
 * Recursive function that tries to insert the new key in
 * in the prvided node, or splits it and go deeper
 * in the BTree hierarchy.
 */
InMemoryBTree.Tree.prototype._insertNonFull = function(node,key,data) {
    var idx = node.numberActives - 1;

    while(!node.isLeaf) {
        while(idx>=0 && this.comparator(key,node.keys[idx].key) === -1) {
            idx--;
        }
        idx++;
        var child = this._diskRead(node.children[idx]);

        if(child.numberActives === 2*this.order -1) {
            this._splitChild(node,idx,child);
            if(this.comparator(key, node.keys[idx].key)===1) {
                idx++;
            }
        }
        node = this._diskRead(node.children[idx]);
        idx = node.numberActives -1;
    }

    while(idx>=0 && this.comparator(key,node.keys[idx].key) === -1) {
        node.keys[idx+1] = node.keys[idx];
        idx--;
    }

    node.keys[idx + 1] = {key:key, data:data};
    node.numberActives++;
    this._diskWrite(node);
};

/**
 * delete
 *
 * Deletes the key from the BTree.
 * If the key is not found, an exception is thrown.
 *
 * @param key the key to be deleted
 * @returns true if the key is deleted false otherwise
 */
InMemoryBTree.Tree.prototype.delete = function(key) {
    var node = this.root;
    var parent = null;
```

```
        var searching = true;
        var idx = null;
        var lsibling = null;
        var rsibling = null;
        var shouldContinue = true;

        while(shouldContinue === true) {
            shouldContinue = false;

            while(searching === true) {
                i = 0;

                if(node.numberActives === 0) {
                    return false;
                }

                while(i<node.numberActives && this.comparator(key, node.keys[i].key) === 1) {
                    i++;
                }

                idx = i;

                if(i<node.numberActives && this.comparator(key, node.keys[i].key) === 0) {
                    searching = false;
                }

                if(searching === true) {

                    if(node.isLeaf === true) {
                        return false;
                    }

                    parent = node;
                    node = this._diskRead(node.children[i]);

                    if(node===null) {
                        return false;
                    }

                    if(idx === parent.numberActives) {
                        lsibling = this._diskRead(parent.children[idx-1]);
                        rsibling = null;
                    } else if(idx === 0) {
                        lsibling = null;
                        rsibling = this._diskRead(parent.children[1]);
                    } else {
                        lsibling = this._diskRead(parent.children[idx-1]);
                        rsibling = this._diskRead(parent.children[idx+1]);
                    }


                    if(node.numberActives === (this.order-1) && parent != null) {
                        if(rsibling != null && rsibling.numberActives > (this.order-1)) {
                            // The current node has (t - 1) keys but the right sibling has > (t - 1) keys
                            this._moveKey(parent,i,left);
                        } else if(lsibling != null && lsibling.numberActives > (this.order-1)) {
                            // The current node has (t - 1) keys but the left sibling has > (t - 1) keys
                            this._moveKey(parent,i,right);
                        } else if(lsibling != null && lsibling.numberActives === (this.order-1)) {
                            // The current node has (t - 1) keys but the left sibling has (t - 1) keys
                            node = this._mergeSiblings(parent,i,left);
                        } else if(rsibling != null && rsibling.numberActives === (this.order-1)){
                            // The current node has (t - 1) keys but the left sibling has (t - 1) keys
                            node = this._mergeSiblings(parent,i,right);
                        }
                    }
                }
```

```
            }


            //Case 1 : The node containing the key is found and is the leaf node.
            //Also the leaf node has keys greater than the minimum required.
            //Simply remove the key
            if(node.isLeaf && (node.numberActives > (this.order-1))) {
                this._deleteKeyFromNode(node,idx);
                return true;
            }


            //If the leaf node is the root permit deletion even if the number of keys is
            //less than (t - 1)
            if(node.isLeaf && (node === this.root)) {
                this._deleteKeyFromNode(node,idx);
                return true;
            }


            //Case 2: The node containing the key is found and is an internal node
            if(node.isLeaf === false) {
                var tmpNode = null;
                var tmpNode2 = null;
                if((tmpNode=this._diskRead(node.children[idx])).numberActives > (this.order-1)) {
                    var subNodeIdx = this._getMaxKeyPos(tmpNode);
                    key = subNodeIdx.node.keys[subNodeIdx.index];

                    node.keys[idx] = key;

                    //this._delete(node.children[idx],key.key);
                    this._diskWrite(node);
                    node = tmpNode;
                    key = key.key;
                    shouldContinue = true;
                    searching = true;
                } else if ((tmpNode = this._diskRead(node.children[idx+1])).numberActives >(this.order-1))
{
                    var subNodeIdx = this._getMinKeyPos(tmpNode);
                    key = subNodeIdx.node.keys[subNodeIdx.index];

                    node.keys[idx] = key;

                    //this._delete(node.children[idx+1],key.key);
                    this._diskWrite(node);
                    node = tmpNode;
                    key = key.key;
                    shouldContinue = true;
                    searching = true;
                } else if((tmpNode = this._diskRead(node.children[idx])).numberActives === (this.order-1)
&&
                    (tmpNode2 = this._diskRead(node.children[idx+1])).numberActives === (this.order-1)) {

                    var combNode = this._mergeNodes(tmpNode, node.keys[idx], tmpNode2);
                    node.children[idx] = combNode;

                    idx++;
                    for(var i=idx; i<node.numberActives; i++) {
                        node.children[i] = node.children[i+1];
                        node.keys[i-1] = node.keys[i];
                    }
                    // freeing unused references
                    node.children[i] = null;
                    node.keys[i-1] = null;

                    node.numberActives--;
                    if (node.numberActives === 0 && this.root === node) {
```

```
                    this.root = combNode;
                }

                this._diskWrite(node);

                node = combNode;
                shouldContinue = true;
                searching = true;
            }
        }


        // Case 3:
        // In this case start from the top of the tree and continue
        // moving to the leaf node making sure that each node that
        // we encounter on the way has atleast 't' (order of the tree)
        // keys
        if(node.isLeaf && (node.numberActives > this.order - 1) && searching===false) {
            this._deleteKeyFromNode(node,idx);
        }

        if(shouldContinue === false) {
            return true;
        }
    }
};

/**
 * _moveKey
 *
 * Move key situated at position i of the parent node
 * to the left or right child at positions i-1 and i+1
 * according to the provided position
 *
 * @param parent the node whose is going to be moved to a child
 * @param i Index of the key in the parent
 * @param position left, or right
 */
InMemoryBTree.Tree.prototype._moveKey = function (parent, i, position) {

    if (position === right) {
        i--;
    }

    //var lchild = parent.children[i-1];
    var lchild = this._diskRead(parent.children[i]);
    var rchild = this._diskRead(parent.children[i + 1]);


    if (position == left) {
        lchild.keys[lchild.numberActives] = parent.keys[i];
        lchild.children[lchild.numberActives + 1] = rchild.children[0];
        rchild.children[0] = null;
        lchild.numberActives++;

        parent.keys[i] = rchild.keys[0];

        for (var _i = 1; _i < rchild.numberActives; _i++) {
            rchild.keys[_i - 1] = rchild.keys[_i];
            rchild.children[_i - 1] = rchild.children[_i];
        }
        rchild.children[rchild.numberActives - 1] = rchild.children[rchild.numberActives];
        rchild.numberActives--;
    } else {
        rchild.children[rchild.numberActives + 1] = rchild.children[rchild.numberActives];
        for (var _i = rchild.numberActives; _i > 0; _i--) {
            rchild.children[_i] = rchild.children[_i - 1];
```

```
            rchild.keys[_i] = rchild.keys[_i - 1];
        }
        rchild.keys[0] = null;
        rchild.children[0] = null;

        rchild.children[0] = lchild.children[lchild.numberActives];
        rchild.keys[0] = parent.keys[i];
        rchild.numberActives++;

        lchild.children[lchild.numberActives] = null;
        parent.keys[i] = lchild.keys[lchild.numberActives - 1];
        lchild.keys[lchild.numberActives - 1] = null;
        lchild.numberActives--;
    }

    this._diskWrite(lchild);
    this._diskWrite(rchild);
    this._diskWrite(parent);
};

/**
 * _mergeSiblings
 *
 * Merges two nodes at the left and right of the provided
 * index in the parent node.
 *
 * @param parent the node whose children will be merged
 * @param i Index of the key in the parent pointing to the nodes to merge
 */
InMemoryBTree.Tree.prototype._mergeSiblings = function (parent, index, pos) {
    var i, j;
    var n1, n2;

    if (index === (parent.numberActives)) {
        index--;
        n1 = this._diskRead(parent.children[parent.numberActives - 1]);
        n2 = this._diskRead(parent.children[parent.numberActives]);
    } else {
        n1 = this._diskRead(parent.children[index]);
        n2 = this._diskRead(parent.children[index + 1]);
    }

    //Merge the current node with the left node
    var newNode = this._allocateNode();
    newNode.isLeaf = n1.isLeaf;
    newNode.level = n1.level;

    for (j = 0; j < this.order - 1; j++) {
        newNode.keys[j] = n1.keys[j];
        newNode.children[j] = n1.children[j];
    }

    newNode.keys[this.order - 1] = parent.keys[index];
    newNode.children[this.order - 1] = n1.children[this.order - 1];

    for (j = 0; j < this.order - 1; j++) {
        newNode.keys[j + this.order] = n2.keys[j];
        newNode.children[j + this.order] = n2.children[j];
    }
    newNode.children[2 * this.order - 1] = n2.children[this.order - 1];

    parent.children[index] = newNode;

    for (j = index; j < parent.numberActives; j++) {
        parent.keys[j] = parent.keys[j + 1];
        parent.children[j + 1] = parent.children[j + 2];
    }
```

```
    newNode.numberActives = n1.numberActives + n2.numberActives + 1;
    parent.numberActives--;

    for (i = parent.numberActives; i < 2 * this.order - 1; i++) {
        parent.keys[i] = null;
    }

    if (parent.numberActives === 0 && this.root === parent) {
        this.root = newNode;
        if (newNode.level) {
            newNode.isLeaf = false;
        } else {
            newNode.isLeaf = true;
        }
    }

    this._diskWrite(newNode);
    if (this.root === newNode) {
        this._updateRootNode(this.root);
    }
    this._diskWrite(parent);
    this._diskDelete(n1);
    this._diskDelete(n2);

    return newNode;
};

/**
 * _deleteKeyFromNode
 *
 * Deletes the key at position index from the provided node.
 *
 * @param node The node where the key will be deleted.
 * @param index The index of the key that will be deletd.
 * @return true if the key can be deleted, false otherwise
 */
InMemoryBTree.Tree.prototype._deleteKeyFromNode = function (node, index) {
    var keysMax = (2 * this.order) - 1;
    if (node.numberActives < keysMax) {
        keysMax = node.numberActives;
    }
    ;

    var i;

    if (node.isLeaf === false) {
        return false;
    }

    var key = node.keys[index];

    for (i = index; i < keysMax - 1; i++) {
        node.keys[i] = node.keys[i + 1];
    }

    // cleaning invalid reference
    node.keys.pop();

    node.numberActives--;

    this._diskWrite(node);

    return true;
};

InMemoryBTree.Tree.prototype._mergeNodes = function (n1, key, n2) {
```

```
        var newNode;
        var i;

        newNode = this._allocateNode();
        newNode.isLeaf = true;

        for (i = 0; i < n1.numberActives; i++) {
            newNode.keys[i] = n1.keys[i];
            newNode.children[i] = n1.children[i];
        }
        newNode.children[n1.numberActives] = n1.children[n1.numberActives];
        newNode.keys[n1.numberActives] = key;

        for (i = 0; i < n2.numberActives; i++) {
            newNode.keys[i + n1.numberActives + 1] = n2.keys[i];
            newNode.children[i + n1.numberActives + 1] = n2.children[i];
        }
        newNode.children[(2 * this.order) - 1] = n2.children[n2.numberActives];

        newNode.numberActives = n1.numberActives + n2.numberActives + 1;
        newNode.isLeaf = n1.isLeaf;
        newNode.level = n1.level;


        this._diskWrite(newNode);
        // @todo
        // delte old nodes from disk
        return newNode;
    };

    /**
     * audit
     *
     * Checks that the tree data structure is
     * valid.
     */
    InMemoryBTree.Tree.prototype.audit = function (showOutput) {
        var errors = [];
        var alreadySeen = [];
        var that = this;

        var foundInArray = function (data) {
            for (var i = 0; i < alreadySeen.length; i++) {
                if (that.comparator(alreadySeen[i], data) === 0) {
                    var error = " !!! duplicated key " + data;
                    if (showOutput === true) {
                        //console.log(error);
                    }
                    errors.push(error);
                }
            }
        };

        var length = null;
        var that = this;
        this.walkNodes(function (n) {
            if (showOutput === true) {
                //console.log("--- Node at " + n.level + " level");
                //console.log(" - leaf? " + n.isLeaf);
                //console.log(" - num actives? " + n.numberActives);
                //console.log(" - keys: ");
            }
            for (var i = n.numberActives; i < n.keys.length; i++) {
                if (n.keys[i] != null) {
                    if (showOutput === true) {
                        //console.log(" * warning : redundant key data");
                        errors.push(" * warning : redundant key data");
```

```
                }
            }
        }

        for (var i = n.numberActives + 1; i < n.children.length; i++) {
            if (n.children[i] != null) {
                if (showOutput === true) {
                    //console.log(" * warning : redundant children data");
                    errors.push(" * warning : redundant key data");
                }
            }
        }


        if (n.isLeaf === false) {
            for (var i = 0; i < n.numberActives; i++) {
                var maxLeft = that._diskRead(n.children[i]).keys[that._diskRead(n.children
[i]).numberActives - 1 ].key;
                var minRight = that._diskRead(n.children[i + 1]).keys[0].key;
                if (showOutput === true) {
                    //console.log("   " + n.keys[i].key + "(" + maxLeft + "," + minRight + ")");
                }
                if (that.comparator(n.keys[i].key, maxLeft) === -1) {
                    var error = " !!! value max left " + maxLeft + " > key " + n.keys[i].key;
                    if (showOutput === true) {
                        //console.log(error);
                    }
                    errors.push(error);
                }
                if (that.comparator(n.keys[i].key, minRight) === 1) {
                    var error = " !!! value min right " + minRight + " < key " + n.keys[i].key;
                    if (showOutput === true) {
                        //console.log(error);
                    }
                    errors.push(error);
                }

                foundInArray(n.keys[i].key);
                alreadySeen.push(n.keys[i].key);
            }
        } else {
            if (length === null) {
                length = n.level;
            } else {
                if (length != n.level) {
                    var error = " !!! Leaf node with wrong level value";
                    if (showOutput === true) {
                        //console.log(error);
                    }
                    errors.push(error);
                }
            }
            for (var i = 0; i < n.numberActives; i++) {
                if (showOutput === true) {
                    //console.log(" " + n.keys[i].key);
                }
                foundInArray(n.keys[i].key);
                alreadySeen.push(n.keys[i].key);

            }
        }

        if (n != that.root) {
            if (n.numberActives > ((2 * that.order) - 1)) {
                if (showOutput === true) {
                    var error = " !!!! MAX num keys restriction violated ";
                }
```

```
                //console.log(error);
                errors.push(error);
            }
            if (n.numberActives < (that.order - 1)) {
                if (showOutput === true) {
                    var error = " !!!! MIN num keys restriction violated ";
                }
                //console.log(error);
                errors.push(error);
            }

        }
    });

    return errors;
};

/**
 *  _getMaxKeyPos
 *
 *  Used to get the position of the MAX key within the subtree
 *  @return An object containing the key and position of the key
 */
InMemoryBTree.Tree.prototype._getMaxKeyPos = function (node) {
    var node_pos = {};

    while (true) {
        if (node === null) {
            break;
        }

        if (node.isLeaf === true) {
            node_pos.node = node;
            node_pos.index = node.numberActives - 1;
            return node_pos;
        } else {
            node_pos.node = node;
            node_pos.index = node.numberActives - 1;
            node = this._diskRead(node.children[node.numberActives]);
        }
    }

    return node_pos;
};

/**
 *  _getMinKeyPos
 *
 *  Used to get the position of the MAX key within the subtree
 *  @return An object containing the key and position of the key
 */
InMemoryBTree.Tree.prototype._getMinKeyPos = function (node) {
    var node_pos = {};

    while (true) {
        if (node === null) {
            break;
        }

        if (node.isLeaf === true) {
            node_pos.node = node;
            node_pos.index = 0;
            return node_pos;
        } else {
            node_pos.node = node;
            node_pos.index = 0;
            node = this._diskRead(node.children[0]);
```

```
            }
        }

        return node_pos;
    };


    /**
     * Node
     *
     * Implements the interface of BinarySearchTree.Node
     *
     * A Tree node augmented with BTree
     * node structures
     */
    InMemoryBTree.Node = function() {
        this.numberActives = 0;
        this.isLeaf = null;
        this.keys = [];
        this.children = [];
        this.level = 0;
    };
    return InMemoryBTree;
});/*global define */
define([], function () {
    var Utils = {};
    Utils.extends = function(supertype, descendant) {
        descendant.prototype = new supertype();
    };

    Utils.stackCounterLimit = 1000;
    Utils.stackCounter = 0;

    Utils.recur = function(c){
        if (Utils.stackCounter === Utils.stackCounterLimit) {
            Utils.stackCounter = 0;
            setTimeout(c, 0);
        } else {
            Utils.stackCounter++;
            c();
        }
    };

    Utils.clone = function(o) {
        return JSON.parse(JSON.stringify(o));
    };

    Utils.shuffle = function(o){ //v1.0
        for(var j, x, i = o.length; i; j = parseInt(Math.random() * i), x = o[--i], o[i] = o[j], o[j] = x)
{}
        return o;
    };

    Utils.include = function(a,v) {
        var cmp = arguments[2];

        for(var i=(a.length-1); i>=0; i--) {
            var res = false;
            if (cmp == null) {
                res = (a[i] === v);
            } else {
                res = (cmp(a[i],v) === 0);
            }

            if (res === true) {
                return true;
            }
```

```
    }

    return false;
};

Utils.remove = function(a,v) {
    var acum = [];
    for(var i=0; i<a.length; i++) {
        if (a[i] !== v) {
            acum.push(a[i]);
        }
    }

    return acum;
};

Utils.repeat = function(c,max,floop,fend,env) {
    if (arguments.length===4) { env = {}; }
    if (c<max) {
        env._i = c;
        floop(function(floop,env){
            // avoid stack overflow
            // deadly hack
            Utils.recur(function(){ Utils.repeat(c+1, max, floop, fend, env); });
        },env);
    } else {
        fend(env);
    }
};


Utils.meanwhile = function(c,floop,fend,env) {
    if (arguments.length===3) { env = {}; }

    if (env['_stack_counter'] == null) {
        env['_stack_counter'] = 0;
    }

    if (c===true) {
        floop(function(c,floop,env){
            if (env['_stack_counter'] % 40 == 39) {
                env['_stack_counter'] = env['_stack_counter'] + 1;
                setTimeout(function(){ Utils.neanwhile(c, floop, fend, env); }, 0);
            } else {
                env['_stack_counter'] = env['_stack_counter'] + 1;
                Utils.meanwhile(c, floop, fend, env);
            }
        },env);
    } else {
        fend(env);
    }
};

Utils.seq = function() {
    var fs = arguments;
    return function(callback) {
        Utils.repeat(0, fs.length, function(k,env){
            var floop = arguments.callee;
            fs[env._i](function(){
                k(floop, env);
            });
        }, function(){
            callback();
        });
    };
};
Utils.partition = function(c, n) {
```

```
        var rem = c.length % n;
        var currentGroup = [];
        var i;
        for(i=0; i<rem; i++) {
            currentGroup.push(null);
        }
        var groups = [];
        for(i=0; i<c.length; i++) {
            currentGroup.push(c[i]);
            if (currentGroup.length % n == 0) {
                groups.push(currentGroup);
                currentGroup = [];
            }
        }
        return groups;
    };
    Utils.keys = function(obj) {
        var variables = [];
        var variable;
        for(variable in obj) {
            if (obj.hasOwnProperty(variable)) {
                variables.push(variable);
            }
        }
        return variables;
    };
    Utils.iso8601 = function(date) {
        function pad(n){
            return n<10 ? '0'+n : n;
        }
        return date.getUTCFullYear()+'-'
            + pad(date.getUTCMonth()+1)+'-'
            + pad(date.getUTCDate())+'T'
            + pad(date.getUTCHours())+':'
            + pad(date.getUTCMinutes())+':'
            + pad(date.getUTCSeconds())+'Z';
    };


    Utils.parseStrictISO8601 = function (str) {
        var regexp = "([0-9]{4})(-([0-9]{2})(-([0-9]{2})" +
            "(T([0-9]{2}):([0-9]{2})(:([0-9]{2})(\.([0-9]+))?)?" +
            "(Z|(([-+])([0-9]{2}):([0-9]{2})))?)?)?)?";
        var d = str.match(new RegExp(regexp));

        var offset = 0;
        var date = new Date(d[1], 0, 1);

        if (d[3]) {
            date.setMonth(d[3] - 1);
        } else {
            throw "missing ISO8061 component"
        }
        if (d[5]) {
            date.setDate(d[5]);
        } else {
            throw "missing ISO8061 component"
        }
        if (d[7]) {
            date.setHours(d[7]);
        } else {
            throw "missing ISO8061 component"
        }
        if (d[8]) {
            date.setMinutes(d[8]);
        } else {
            throw "missing ISO8061 component"
```

```
        }
        if (d[10]) {
            date.setSeconds(d[10]);
        } else {
            throw "missing ISO8061 component"
        }
        if (d[12]) {
            date.setMilliseconds(Number("0." + d[12]) * 1000);
        }
        if (d[14]) {
            offset = (Number(d[16]) * 60) + Number(d[17]);
            offset *= ((d[15] == '-') ? 1 : -1);
        }

        offset -= date.getTimezoneOffset();
        var time = (Number(date) + (offset * 60 * 1000));
        var toReturn = new Date();
        toReturn.setTime(Number(time));
        return toReturn;
    };


    Utils.parseISO8601 = function (str) {
        var regexp = "([0-9]{4})(-([0-9]{2})(-([0-9]{2})" +
            "(T([0-9]{2}):([0-9]{2})(:([0-9]{2})(\.([0-9]+))?)?" +
            "(Z|(([-+])([0-9]{2}):([0-9]{2})))?)?)?)?";
        var d = str.match(new RegExp(regexp));

        var offset = 0;
        var date = new Date(d[1], 0, 1);

        if (d[3]) { date.setMonth(d[3] - 1); }
        if (d[5]) { date.setDate(d[5]);  }
        if (d[7]) { date.setHours(d[7]);  }
        if (d[8]) { date.setMinutes(d[8]);  }
        if (d[10]) { date.setSeconds(d[10]);  }
        if (d[12]) { date.setMilliseconds(Number("0." + d[12]) * 1000); }
        if (d[14]) {
            offset = (Number(d[16]) * 60) + Number(d[17]);
            offset *= ((d[15] == '-') ? 1 : -1);
        }

        offset -= date.getTimezoneOffset();
        var time = (Number(date) + (offset * 60 * 1000));
        var toReturn = new Date();
        toReturn.setTime(Number(time));
        return toReturn;
    };

    Utils.parseISO8601Components = function (str) {
        var regexp = "([0-9]{4})(-([0-9]{2}))(-([0-9]{2}))(T([0-9]{2}):([0-9]{2})(:([0-9]{2}))?(\.([0-9]
+))?)?(Z|([-+])([0-9]{2}):([0-9]{2}))?)?";
        var d = str.match(new RegExp(regexp));
        var year, month, date, hours, minutes, seconds, millisecs, timezone;
        year = Number(d[1]);
        month = d[3] - 1;
        date  = Number(d[5]);
        hours = Number(d[7]);
        minutes = Number(d[8]);
        seconds = Number(d[10]);

        if (d[12]) { millisecs = Number("0." + d[12]) * 1000; }

        if (d[13]==="Z") {
            timezone = 0;
        } else if (d[14]) {
            timezone = 0;
```

```
            if (d[17]) {
                timezone = Number(d[17]);
            }
            timezone = timezone+(Number(d[15]) * 60);
            timezone *= ((d[14] == '-') ? -1 : +1);
        } else if (d[14]==null && d[11]) {
            timezone = Number(d[12])*60;
        }

        return {'year': isNaN(year) ? null : year,
            'month': isNaN(month) ? null : month,
            'date': isNaN(date) ? null : date,
            'hours': isNaN(hours) ? null : hours,
            'minutes': isNaN(minutes) ? null : minutes,
            'seconds': isNaN(seconds) ? null : seconds,
            'millisecs':isNaN(millisecs) ? null : millisecs,
            'timezone': isNaN(timezone) ? null : timezone};
    };

    Utils.compareDateComponents = function(stra,strb) {
        var a = Utils.parseISO8601Components(stra);
        var b = Utils.parseISO8601Components(strb);
        var offset;
        if ((a.timezone == null && b.timezone == null) ||
            (a.timezone != null && b.timezone != null)) {
            var da = Utils.parseISO8601(stra);
            var db = Utils.parseISO8601(strb);

            if (da.getTime() == db.getTime()) {
                return 0;
            } else if (da.getTime() < db.getTime()){
                return -1;
            } else {
                return 1;
            }
        } else if (a.timezone != null && b.timezone == null){
            da = Utils.parseISO8601(stra);
            db = Utils.parseISO8601(strb);
            var ta = da.getTime();
            var tb = db.getTime();
            offset = 14*60*60;
            if (ta < tb && ta < (tb + offset)) {
                return -1;
            } else if (ta > tb && ta > (tb - offset)) {
                return 1;
            } else {
                return null;
            }
        } else {
            da = Utils.parseISO8601(stra);
            db = Utils.parseISO8601(strb);
            ta = da.getTime();
            tb = db.getTime();
            offset = 14*60*60;
            if (ta < tb && (ta + offset)  < tb) {
                return -1;
            } else if (ta > tb && (ta + offset) > tb) {
                return 1;
            } else {
                return null;
            }
        }
    };
    // RDF utils
    Utils.lexicalFormLiteral = function(term, env) {
        var value = term.value;
        var lang = term.lang;
```

```
        var type = term.type;

        var indexedValue = null;
        if (value != null && type != null && typeof(type) != 'string') {
            var typeValue = type.value;

            if (typeValue == null) {
                var typePrefix = type.prefix;
                var typeSuffix = type.suffix;

                var resolvedPrefix = env.namespaces[typePrefix];
                term.type = resolvedPrefix+typeSuffix;
                typeValue = resolvedPrefix+typeSuffix;
            }
            // normalization
            if (typeValue.indexOf('hexBinary') != -1) {
                indexedValue = '"' + term.value.toLowerCase() + '"^^<' + typeValue + '>';
            } else {
                indexedValue = '"' + term.value + '"^^<' + typeValue + '>';
            }
        } else {
            if (lang == null && type == null) {
                indexedValue = '"' + value + '"';
            } else if (type == null) {
                indexedValue = '"' + value + '"' + "@" + lang;
            } else {
                // normalization
                if (type.indexOf('hexBinary') != -1) {
                    indexedValue = '"' + term.value.toLowerCase() + '"^^<'+type+'>';
                } else {
                    indexedValue = '"' + term.value + '"^^<'+type+'>';
                }
            }
        }
        return indexedValue;
    };

    Utils.lexicalFormBaseUri = function(term, env) {
        var uri = null;
        //console.log("*** normalizing URI token:");
        //console.log(term);
        if (term.value == null) {
            //console.log(" - URI has prefix and suffix");
            //console.log(" - prefix:"+term.prefix);
            //console.log(" - suffixx:"+term.suffix);
            var prefix = term.prefix;
            var suffix = term.suffix;
            var resolvedPrefix = env.namespaces[prefix];
            if (resolvedPrefix != null) {
                uri = resolvedPrefix+suffix;
            } else {
                uri = prefix+":"+suffix;
            }
        } else {
            //console.log(" - URI is not prefixed");
            uri = term.value;
        }

        if (uri===null) {
            return null;
        } else {
            //console.log(" - resolved URI is "+uri);
            if (uri.indexOf(":") == -1) {
                //console.log(" - URI is partial");
                uri = (env.base||"") + uri; // applyBaseUri
            } else {
                //console.log(" - URI is complete");
```

```
            }
            //console.log(" -> FINAL URI: "+uri);
        }

        return uri;
    };


    Utils.lexicalFormTerm = function(term, ns) {
        if (term.token === 'uri') {
            return {'uri': Utils.lexicalFormBaseUri(term, ns)};
        } else if (term.token === 'literal') {
            return {'literal': Utils.lexicalFormLiteral(term, ns)};
        } else if (term.token === 'blank') {
            var label = '_:'+term.value;
            return {'blank': label};
        } else {
            throw "Error, cannot get lexical form of unknown token: "+term.token;
        }
    };
    Utils.normalizeUnicodeLiterals = function (string) {
        var escapedUnicode = string.match(/\\u[0-9abcdefABCDEF]{4,4}/g) || [];
        var dups = {};
        for (var i = 0; i < escapedUnicode.length; i++) {
            if (dups[escapedUnicode[i]] == null) {
                dups[escapedUnicode[i]] = true;
                string = string.replace(new RegExp("\\" + escapedUnicode[i], "g"), eval("'" +
escapedUnicode[i] + "'"));
            }
        }

        return string;
    };
    Utils.hashTerm = function(term) {
        try {
            if (term == null) {
                return "";
            }
            if (term.token==='uri') {
                return "u"+term.value;
            } else if (term.token === 'blank') {
                return "b"+term.value;
            } else if (term.token === 'literal') {
                var l = "l"+term.value;
                l = l + (term.type || "");
                l = l + (term.lang || "");

                return l;
            }
        } catch(e) {
            if (typeof(term) === 'object') {
                var key = "";
                Utils.each(term, function (t, p) {
                    key = key + p + t;
                });
                return key;
            }
            return term;
        }
    };
    return Utils;
});
```