ÿþ<!DOCTYPE html> <html lang=en style><!-- Page saved with SingleFile url: file:///C:/src/word/HTML2Doc/shex/index.html saved date: Tue May 27 2025 10:51:57 GMT+0200 (hora de verano de Europa central) --><meta charset=utf-8> <meta name=viewport content="width=device-width"> <title>Shape Expressions Language 2.next</title> <style>h2,h3{color:#000!important}</style> <meta name=referrer content=no-referrer><link type=image/x-icon rel="shortcut icon" href=data:application/octet-stream;base64,><meta http-equiv=content-security-policy content="default-src 'none'; font-src 'self' data:; img-src 'self' data:; style-src 'unsafe-inline'; media-src 'self' data:; script-src 'unsafe-inline' data:; object-src 'self' data:; frame-src 'self' data:;"></head> <body> <section id=abstract> <p> The Shape Expressions (ShEx) language describes <a>RDF nodes</a> and <a>graph</a> structures. A <span class=math>node constraint</span> describes an RDF node (<a>IRI</a>, <a>blank node</a> or <a>literal</a>) and a <span class=math>shape</span> describes the <a>triples</a> involving nodes in an <a>RDF graph</a>. These descriptions identify <a data-lt=predicate>predicates</a> and their associated cardinalities and datatypes. ShEx shapes can be used to communicate data structures associated with some process or interface, generate or validate data, or drive user interfaces. </p> <p> This document defines the ShEx language. See the <a href=file:///C:/src/word/HTML2Doc/primer/index.html>Shape Expressions Primer</a> for a non-normative description of ShEx. </p> </section> <section id=keywords> <p> RDF, Schema, Shape Expressions, Structure Definition, Structural Validation </p> </section> <section id=sotd> <p> This is an editor's draft of the Shape Expressions specification. ShEx 2.x differs significantly from the W3C ShEx Submission. The <a href=http://shex.io/shex-semantics-20170713/>July 2017 publication</a> included a <a href=#validation>definition of validation</a> which implied infinite recursion. This version explicitly includes recursion checks. No tests changed as a result of this and no implementations or applications are known to have been affected. </p> <p> If you wish to make comments regarding this document, please raise them as GitHub issues. There are separate interfaces for <a href=https://github.com/shexSpec/spec/issues>specification</a>, <a href=https://github.com/shexSpec/shex/issues>language</a> and <a href=https://github.com/shexSpec/shexTest/issues>test</a> issues. Only send comments to <a href=mailto:public-shex@w3.org>public-shex@w3.org</a> (<a href="mailto:public-shex-request@w3.org?subject=subscribe">subscribe</a>, <a href=https://lists.w3.org/Archives/Public/public-shex/>archives</a>) if you are unable to raise issues on GitHub. All comments are welcome. </p> </section> <section id=conformance> <h2>Conformance</h2> <p>Conformance criteria are relevant to authors and authoring tool implementers. As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.</p> <p> All ShEx documents MUST conform to the <a href=#schema-requirements class=sectionRef>Schema Requirements</a>. Additional constraints for the specific types of ShEx documents (<a>ShExC</a>, <a>ShExJ</a>, and <a>ShExR</a>) follow: </p> <ul> <li>A <a>ShExC</a> document is a UTF-8 document which conforms to the grammar described in <a href=#shexc class=sectionRef></a> resulting in a valid <a>ShExJ</a> document.</li> <li>A <a>ShExJ</a> document is a valid JSON-LD document [[!JSON-LD]], and conforms to the <a>ShExJ</a> syntax, as described in <a href=#shexj class=sectionRef></a>.</li> <li>JSON documents can be interpreted as <a>ShExJ</a> by following the normative statements in <a data-cite=JSON-LD#interpreting-json-as-json-ld>Section 4.8 Interpreting JSON as JSON-LD</a> in [[!JSON-LD]].</li> A <a>ShExR</a> RDF document complies with this specification if it conforms to the schema in <a href=#shexr class=sectionRef>ShExR.shex</a>. </ul> </section> <section id=introduction> <h2>Introduction</h2> <p> The Shape Expressions (<dfn>ShEx</dfn>) language provides a structural schema for RDF data. This can be used to document APIs or datasets, aid in development of API-conformant messages, minimize defensive programming, guide user interfaces, or anything else that involves a machine-readable description of data organization and typing requirements. </p> <p> ShEx describes <a>RDF graph</a> [[RDF11-CONCEPTS]] structures as sets of potentially connected <a data-lt=shape>Shapes</a>. These constrain the <a>triples</a> involving nodes in an <a>RDF graph</a>. <code>Node Constraints</code> constrain RDF nodes by constraining their node kind (<a>IRI</a>, <a>blank node</a> or <a>Literal</a>), enumerating permissible values in value sets, specifying their datatype, and constraining value ranges of Literals. Additionally, they constrain lexical forms of <a>Literals</a>, <a>IRIs</a> and <a>labeled blank nodes</a>. Shape Expressions schemas share blank nodes with the constrained <a>RDF graphs</a> in the same way that graphs in <a data-cite=rdf11-concepts#dfn-rdf-dataset>RDF datasets</a> [[!rdf11-concepts]] share blank nodes. </p> <p> ShEx can be represented in JSON structures (<a>ShExJ</a>) or a compact syntax (<a>ShExC</a>). The compact syntax is intended for human consumption; the JSON structure for machine processing. This document defines ShEx in terms of <a>ShExJ</a> and includes a <a href=#shexc>section on the ShEx Compact Syntax (ShEx)</a>. </p> </section> <section id=defn-acro-abbrs> <h2>Definitions, Acronyms, and Abbreviations</h2> <section id=definitions> <h3>Definitions</h3> <p>Shape expressions are defined using terms from RDF semantics [[!rdf11-mt]]:</p> <ul> <li><dfn data-cite=rdf11-concepts#dfn-node data-lt="nodes|rdf node|rdf nodes">Node</dfn>: one of <dfn data-cite=rdf11-concepts#dfn-iri data-lt=iris>IRI</dfn>, <dfn data-cite=rdf11-concepts#dfn-blank-node data-lt="bnodes|blank nodes">blank node</dfn>, <dfn data-cite=rdf11-concepts#dfn-literal data-lt="literal|literals|RDF Literal|RDF Literals">Literal</dfn></li> <li><dfn data-cite=rdf11-concepts#dfn-rdf-graph data-lt="RDF Graph|RDF Graphs">Graph</dfn>: a set of <dfn data-cite=rdf11-concepts#dfn-rdf-triple data-lt="triple|rdf triple|rdf triples">Triples</dfn> of (<dfn data-cite=rdf11-concepts#dfn-subject>subject</dfn>, <dfn data-cite=rdf11-concepts#dfn-predicate>predicate</dfn>, <dfn data-cite=rdf11-concepts#dfn-object>object</dfn>)</li> </ul> <p> The following functions access the elements of an <a>RDF graph</a> <span class=math>G</span> containing a node <span class=math>n</span>:</p> <ul> <li> <span class=math><dfn>arcsOut</dfn>(G, n)</span> is the set of <a>triples</a> in a <a>graph</a> <span class=math>G</span> with <a>subject</a> <span class=math>n</span>. </li> <li> <span class=math><dfn>predicatesOut</dfn>(G, n)</span> is the set of <a data-lt=predicate>predicates</a> in <span class=math><a>arcsOut</a>(G, n)</span>. </li> <li> <span class=math><dfn>arcsIn</dfn>(G, n)</span> is the set of <a>triples</a> in a <a>graph</a> <span class=math>G</span> with <a>object</a> <span class=math>n</span>. </li> <li> <span class=math><dfn>predicatesIn</dfn>(G, n)</span> is the set of <a data-lt=predicate>predicates</a> in <span class=math><a>arcsIn</a>(G, n)</span>. </li> <li> <span class=math><dfn>neigh</dfn>(G, n)</span> is the neighbourhood of the <a>node</a> <span class=math>n</span> in the <a>graph</a> <span class=math>G</span>.<br><span class=math><a>neigh</a>(G, n) = <a>arcsOut</a>(G, n) Ôê¬ <a>arcsIn</a>(G, n)</span>. </li> <li> <span class=math><dfn>predicates</dfn>(G, n)</span> is the set of <a data-lt=predicate>predicates</a> in <span class=math><a>neigh</a>(G, n)</span>.<br><span class=math><a>predicates</a>(G, n) = <a>predicatesOut</a>(G, n) Ôê¬ <a>predicatesIn</a>(G, n)</span>. </li> <li> <span class=math><dfn>def</dfn>(Sch, label)</span> is the <span class=math>decl</span>.<span class=param>shapeExpr</span> where <span class=math>decl</span>.<span class=param>label</span> = <code>label</code>. <code>Sch</code> must have exactly one <code>def(Sch, label)</code>. </li> </ul> <div class=example> <p> Consider the <a>RDF graph</a> <span class=math>G</span> represented in Turtle: </p> <pre class=data>PREFIX ex: http://schema.example/# PREFIX inst: http://inst.example/# PREFIX foaf: http://xmlns.com/foaf/ PREFIX xsd: http://www.w3.org/2001/XMLSchema# inst:Issue1 ex:state ex:unassigned ; ex:reportedBy \_:User2 . \_:User2 foaf:name "Bob Smith" ; foaf:mbox &lt;mailto:bob@example.org&gt; . </pre> <p> There are two arcs out of <span class=pre>\_:User2</span>; <span class=math><a>arcsOut</a>(G, <span class=pre>\_:User2</span>)</span>: </p> <pre class=data> \_:User2 foaf:name "Bob Smith" . \_:User2 foaf:mbox &lt;mailto:bob@example.org&gt; . </pre> <p> There is one arc into <span class=pre>\_:User2</span>; <span class=math><a>arcsIn</a>(G, <span class=pre>\_:User2</span>)</span>: </p> <pre class=data> inst:Issue1 ex:reportedBy \_:User2 . </pre> <p> There are three arcs in the neighbourhood of <span class=pre>\_:User2</span> set, <span class=math><a>neigh</a>(G, <span class=pre>\_:User2</span>)</span>: </p> <pre class=data> \_:User2 foaf:name "Bob Smith" . \_:User2 foaf:mbox &lt;mailto:bob@example.org&gt; . inst:Issue1 ex:reportedBy \_:User2 . </pre> </div> </section> <section id="Acronyms and abbreviations"> <h3>Acronyms And Abbreviations</h3> <table id=acronyms class=bare> <tbody> <tr> <td>BNF</td> <td>Backus Naur Form</td> </tr> <tr> <td>CSS</td> <td>Cascading Stylesheets</td> </tr> <tr> <td>IANA</td> <td>Internet Assigned Numbers Authority</td> </tr> <tr> <td>IRI</td> <td>Internationalized Resource Identifier</td> </tr> <tr> <td>RDF</td> <td>Resource Description Framework</td> </tr> <tr> <td>ShEx</td> <td>Shape Expressions RDF schema language</td> </tr> <tr> <td>ShExC</td> <td>ShEx Compact syntax</td> </tr> <tr> <td>ShExJ</td> <td>ShEx JSON (or JSON-LD) syntax</td> </tr> <tr> <td>ShExR</td> <td>ShEx RDF syntax</td> </tr> <tr> <td>SPARQL</td> <td>RDF Query Language</td> </tr> <tr> <td>URL</td> <td>Uniform Resource Locator</td> </tr> <tr> <td>UTF-8</td> <td>Unicode Transformation Format</td> </tr> <tr> <td>XML</td> <td>Extensible Markup Language</td> </tr> <tr> <td>XPath</td> <td>Path Language for XML</td> </tr> </tbody> </table> </section> </section> <section id=notation> <h2>Notation</h2> <p> The JSON [[!rfc7159]] Syntax serves as a serializable proxy for an abstract syntax. </p> <p> <a data-cite=rdf11-concepts/#dfn-rdf-term>RDF terms</a> are represented as <a data-cite=json-ld#dfn-node>JSON-LD nodes</a>. <ul> <li><a>IRIs</a> are represented as a <a data-cite=rfc7159#section-7>JSON string</a> consisting of the IRI string, e.g. <br><code class=json>"http://example.org/resource"</code></li> <li><a>Blank nodes</a> are represented as a JSON string composed of the concatenation of "<code>\_:</code>" and a <dfn data-cite=rdf11-concepts#dfn-blank-node-identifier data-lt="blank node identifiers|labeled blank nodes">blank node identifier</dfn>, e.g. <br><code class=json>"\_:blank3"</code></li> <li><a>Literals</a> are represented as a <a>JSON object</a>s following the composition rules for <a data-cite=json-ld/#dfn-json-ld-value>JSON-LD values</a>, i.e. <ul> <li>literals with the datatype <code>http://www.w3.org/2001/XMLSchema#string</code> are represented with the <span class=param>value</span> property, e.g. <br><code class=json>{ "value": "abc" }</code>.</li> <li><a>language-tagged strings</a> are represented with an additional <span class=param>language</span> property, e.g. <br><code class=json>{ "value": "hello world", "langague": "en-US" }</code></li> <li>datatyped literals are represented with an additional <span class=param>datatype</span> property, e.g. <br><code class=json>{ "value": "123", "datatype": "http://www.w3.org/2001/XMLSchema#integer" }</code> </li> </ul> </li> </ul> <p></p> <section id=json-grammar> <h3>JSON Grammar</h3> <p> This specification uses a <dfn>JSON grammar</dfn> to describe the set of JSON documents that can be interpreted as a ShEx schema. ShEx data structures are represented as <dfn data-cite=rfc7159#section-4 data-lt="json object">JSON objects</dfn> with a member with the name "<code>type</code>" (i.e. an object with a <span class=param>type</span> attribute): </p> <div class=json> <pre class="json nohighlight">{ "type": "typeName", member<sub>0ÔÇªn</sub> }</pre> </div> <p> These are expressed in JSON grammar as <span class=shexj>typeName { member\* }</span>. <a data-cite=rfc7159#section-2>RFC7159 Section 2</a> provides syntactic constraints for JSON ÔÇö the grammar constraining those to valid <a>ShExJ</a> constructs is composed of: </p> <ul> <li><code>typeName</code> is the name of the typed data structure. Types are referenced in the definitions of object members and in the definitions of the semantics for those data structures.</li> <li><code>member\*</code> is a list of zero or more terminals or references to other typeExpressions. </li> <li>A <code>typeExpression</code> is one of: <ul> <li><code>typeName</code> ÔÇö an object of corresponding type</li> <li><code>array</code>: <span class=shexj>[ typeExpression+ ]</span>ÔÇö an array of one or more JSON values matching the typeExpression.</li> <li><code>choice</code>: <span class=shexj>typeExpression1 | typeExpression2 | ÔÇª</span>ÔÇö a choice between two or more typeExressions.</li> </ul> </li> <li>Cardinalities are represented as by the strings <code>?</code>, <code>+</code>, <code>\*</code> following the <a data-cite=XML#sec-notation>notation in the XML specification</a>[[!XML]] or <code>{m,}</code> to indicate a that at least <code>m</code> elements are required.</li> </ul> <p> The following examples are excerpts from the definitions below. In the JSON notation, </p> <div class="shexj simupre"><a class=obj>Schema</a> { <a class=param>startActs</a>:[<a class=objref>SemAct</a>+]? <a class=param>start</a>:<a class=nobref>shapeExpr</a>? <a class=param>imports</a>:[<a>IRI</a>+]? <a class=param>shapes</a>:[<a class=nobref>shapeExpr</a>+]? }</div> <p> signifies that a <code>Schema</code> has four optional components called <a class=param>startActs</a>, <a class=param>start</a>, <a class=param>imports</a> and <a class=param>shapes</a>: </p> <ul> <li><span class=param>startActs</span> is a list of one or more <a class=objref>SemAct</a>.</li> <li><span class=param>start</span> is a <a class=objref>shapeExpr</a>.</li> <li><span class=param>imports</span> is a list of one or more <a class=objref>IRI</a>.</li> <li><span class=param>shapes</span> is an array of <a class=nobref>shapeExpr</a>.</li> </ul> <div class="shexj simupre"><a class=nob>shapeExpr</a> = <a class=objref>ShapeOr</a> | <a class=objref>ShapeAnd</a> | <a class=objref>ShapeNot</a> | <a class=objref>NodeConstraint</a> | <a class=objref>Shape</a> | <a class=objref>ShapeExternal</a> ;</div> <p> signifies that a <a>shapeExpr</a> is one of seven object types: <a class="objref .shexj">ShapeOr</a> | <a class=objref>ShapeAnd</a> | ÔÇª. </p> <div class="shexj simupre"><a class=obj>NodeConstraint</a> { <span class=param>nodeKind</span>:(<span class=literal>"iri"</span> | <span class=literal>"bnode"</span> | <span class=literal>"nonliteral"</span> | <span class=literal>"literal"</span>)? <a class=nobref>xsFacet</a>\* } <a class=nob>xsFacet</a> = <a class=nobref>stringFacet</a> | <a class=nobref>numericFacet</a> ; </div> <p> signifies that a <code>NodeConstraint</code> has a <code class=param>nodeKind</code> of one of the four literals followed by any number of <a class="objref .shexj">xsFacet</a> and an <a class="objref .shexj">xsFacet</a> is either a <a class="objref .shexj">stringFacet</a> or a <a class="objref .shexj">numericFacet</a>. </p> <p class=note> The <a href=https://github.com/shexSpec/shexTest/blob/main/doc/ShExJ.jsg#L11>executable JSON grammar for ShExJ</a> specifically disables the requirement for a matching <code>"type"</code> attribute in <code>ObjectLiteral</code> as <code>"type"</code> is instead used for the datatype of a <a href=https://www.w3.org/TR/json-ld11/#dfn-typed-value>JSON-LD typed value</a>. </p> </section> <section id=shex-references> <h3>References</h3> <p> <a>ShExJ</a> is a dialect of JSON-LD [[!JSON-LD]] and the member <dfn class=param>id</dfn> is used as a <a data-cite=JSON-LD#node-identifiers>node identifier</a>. An ShapeDecl or tripleExpr may be represented inline or referenced by its <span class=param>id</span> which may be either a <a>blank node</a> or an <a>IRI</a>. </p> <div class="shexj simupre"><a class=obj>ShapeOr</a> { <a class=param>id</a>:<a class=nobjref>shapeExprLabel</a>? <a class=param>shapeExprs</a>:[<a class=nobjref>shapeExpr</a>{2,}] } <a class=nob>shapeExprLabel</a> = <a class=nobref>IRIREF</a> | <a class=objref>BNODE</a> ; <a class=obj>EachOf</a> { <a class=param>id</a>:<a class=nobjref>tripleExprLabel</a>? <a class=param>expressions</a>:[<a class=nobjref>tripleExpr</a>{2,}] ... } <a class=nob>tripleExprLabel</a> = <a>IRI</a> | <a>BNODE</a> ; </div> <p> The JSON structure may include references to <a data-lt=shape>shape expressions</a> and <a>triple expressions</a>: </p> <div class="shexj simupre"><a class=obj>shapeExpr</a> = <a class=objref>ShapeOr</a> | ... | <a class=nobjref>shapeExprRef</a> ; <a class=nob>shapeExprRef</a> = <a class=nobjref>shapeExprLabel</a> ; <a class=nobjref>tripleExpr</a> = <a class=objref>EachOf</a> | ... | <a class=nobjref>tripleExprRef</a> ; <a class=nob>tripleExprRef</a> = <a class=objref>tripleExprLabel</a> ; </div> <p> An object with a circular reference must be referenced by an <span class=param>id</span>. This example uses a nested shape reference on a value expression (<a data-lt=TripleConstraint>defined below</a>). </p> <div class=json> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "<span class=lookit>http://schema.example/#IssueShape</span>", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#related", "valueExpr": "<span class=lookit>http://schema.example/#IssueShape</span>", "min": 0 } } ] }</pre> </div> <p> Not captured in this JSON syntax definition is the rule that every <a class=nobjref>shapeExpr</a> nested in a schema's <a class=param>shapes</a> must have an <span class=param>id</span> and no other <a class=nobjref>shapeExpr</a> may have an <span class=param>id</span>. The JSON syntax definitition simplifies this by adding <span style="border:thin solid #888888;background-color:#F8F8F0"><a class=param>id</a>:<a class=nobjref>shapeExprLabel</a>?</span> to every <a class=nobjref>shapeExpr</a>. This example includes a nested shape. Nested shapes are not permitted to have <span class=param>ids</span>. </p> <div class=json> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#submittedBy", "valueExpr": { "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#name", "valueExpr": { "type": "NodeConstraint", "nodeKind": "literal" } } } } } ] }</pre> </div> </section> <section id=style> <h3>Document style</h3> <p> JSON examples are rendered in a <span class=json>.json</span> CSS style. Partial examples include ranges in a <span class=comment>.comment</span> CSS style to indicate text which would be substituted in a complete example. For example <span class=json>{ "type": "ShapeAnd", "shapeExprs": [ <span class=comment>SE<sub>1</sub></span>, <span class=comment>ÔÇª</span> ] }</span> indicates that both <span class=comment>SE<sub>1</sub></span> and <span class=comment>ÔÇª</span> would be substituted in a complete example. </p> <p> In javascript-enabled browsers, schemas with a <button class=show-repchoice>json</button> button can be converted between the JSON representation and the compact syntax by clicking the button. The button text indicates the currently shown representation. Selecting the example and pressing "j" or "c" converts the example to the JSON (ShExJ) or compact form (ShExC). Pressing "shift J" or "shift C" converts all such examples to ShExJ or ShExC. </p> </section> <section id=graphAccess> <h3>Graph access</h3> <p> The validation process defined in this document relies on matching <a>triple patterns</a> in the form <code>(subject, predicate, object)</code> where each position may be supplied by a constant, a previously defined term, or the underscore "<code>\_</code>", which represents a previously undefined element or wildcard. This corresponds to a <dfn data-cite=sparql11-query#defn\_TriplePattern data-lt="triple pattern|triple patterns">SPARQL Triple Pattern</dfn> where each "\_" is replaced by a unique <a>blank node</a>. Matching such a <a>triple pattern</a> against a <a>graph</a> is defined by <a data-cite=sparql11-query#BGPsparql>SPARQL Basic Graph Pattern Matching</a> (BGP) with a BGP containing only that <a>triple pattern</a>. </p> </section> <section id=namespaces> <h3>Namespaces</h3> <p>This specification makes use of the following namespaces:</p> <dl> <dt><code>foaf</code>:</dt> <dd><code>http://xmlns.com/foaf/0.1/</code></dd> <dt><code>rdf</code>:</dt> <dd><code>http://www.w3.org/1999/02/22-rdf-syntax-ns#</code></dd> <dt><code>rdfs</code>:</dt> <dd><code>http://www.w3.org/2000/01/rdf-schema#</code></dd> <dt><code>shex</code>:</dt> <dd><code>http://www.w3.org/ns/shex#</code></dd> <dt><code>xsd</code>:</dt> <dd><code>http://www.w3.org/2001/XMLSchema#</code></dd> </dl> </section> </section> <section id=shapes-language> <h2>The Shape Expressions Language</h2> <p> A Shape Expressions (ShEx) schema is a collection of labeled <a data-lt=shape>Shapes</a> and <a href=#node-constraints>Node Constraints</a>. These can be used to describe or test nodes in <a>RDF graphs</a>. ShEx does not prescribe a language for associating <a>nodes</a> with <a data-lt=shape>shapes</a> but several approaches are <a href=file:///C:/src/word/HTML2Doc/primer/index.html#associating-nodes-with-shapes>described in the ShEx Primer</a>. </p> <section id=shapes-schema> <h3>Shapes Schema</h3> <p> A shapes schema is captured in a <span class=jobjref><dfn data-lt="shapes schema">Schema</dfn></span> object with a list of <span class=jobjref><dfn data-lt=ShapeDecl>Shape Declarationss</dfn></span>: </p> <div class=shexjTable> <table class=shexj> <tbody><tr class=obj> <th class=obj><a>Schema</a></th> <td>{</td> <td><dfn class=param>imports</dfn>:[<a>IRIREF</a>+]? <dfn class=param>startActs</dfn>:[<a class=objref>SemAct</a>+]? <dfn class=param>start</dfn>:<a class=nobref>shapeExpr</a>? <dfn class=param>shapes</dfn>:[<a class=objref>ShapeDecl</a>+]? }</td> </tr> <tr class=obj> <th class=obj><a>ShapeDecl</a></th> <td>{</td> <td> <span class=param>id</span>:<a class=nobref>shapeExprLabel</a> <span class=param>abstract</span>:<a class=nobref>BOOL</a>? <span class=param>shapeExpr</span>:<a class=nobref>shapeExpr</a> | <a class=objref>ShapeExternal</a> }</td> </tr> </table> </div> <p> where <a class=param>shapes</a> is a list of <a>ShapeDecls</a>. </p> <div class="repchoice incomplete"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", <span class=comment>ÔÇª</span> }, { "id": "\_:UserShape", <span class=comment>ÔÇª</span> }, { "id": "http://schema.example/#EmployeeShape", <span class=comment>ÔÇª</span> } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ÔÇª } \_:UserShape { ÔÇª } ex:EmployeeShape { ÔÇª } </pre> </div> </section> <section id=validation> <h3>Validation Definition</h3> <p> For a graph <span class=math>G</span>, a schema <span class=math>Sch</span> and a fixed ShapeMap <span class=math>ism</span>, <code class=function><dfn>isValid</dfn>(<span class=math>G</span>, <span class=math>Sch</span>, <span class=math>ism</span>)</code> indicates that for every shape association <span class=math>(<span class=param>node</span>: n, <span class=param>shape</span>: sl, <span class=param>exact</span>: exact)</span> in <span class=math>ism</span>, the node <span class=math>n</span> satisfies the shape expression identified by <span class=math>sl</span>. If <span class=param>exact</span> is true, the result is <code class=function><a>satisfies</a>(<span class=math>n</span>, <a>def</a>(<span class=math>s.label</span>), <span class=math>G</span>, <span class=math>Sch</span>, <a>completeTyping</a>(<span class=math>G</span>, <span class=math>Sch</span>), <span class=math>neigh(n)</span>)</code> , otherwise, the result is <code class=function><a>satisfiesDescendant</a>(<span class=math>n</span>, <a>def</a>(<span class=math>s.label</span>), <span class=math>G</span>, <span class=math>Sch</span>, <a>completeTyping</a>(<span class=math>G</span>, <span class=math>Sch</span>)</code>, <span class=math>neigh(n)</span>). The function <code class=function><a>satisfies</a></code> is defined for every kind of <a>shape expression</a>. </p> <p> The validation of an RDF graph <span class=math>G</span> against a ShEx schema <span class=math>Sch</span> is based on the existence of <code class=function><a>completeTyping</a>(<span class=math>G</span>, <span class=math>Sch</span>)</code>. For an RDF graph <span class=math>G</span> and a shapes schema <span class=math>Sch</span>, a <dfn>typing</dfn> is a set of pairs of the form <span class=math>(n, l)</span> where <span class=math>n</span> is a node in <span class=math>G</span> and <span class=math>l</span> is a shape label that appears in the shape declarations of the schema. A <dfn label="correct typing">correct typing</dfn> is a <span class=math>typing</span> such that for every RDF node/shape pair <span class=math>(n,l)</span>, satisfies(n, def(l), G, Sch, typing, neigh(n)) holds or satisfiesDescendant(n, def(l), G, Sch, typing, neigh(n)) holds. A <code class=function><a>completeTyping</a>(<span class=math>G</span>, <span class=math>Sch</span>)</code> is a unique correct typing that exists for every graph and every ShEx schema that satisfies the <a href=#schema-requirements>schema requirements</a>. </p> <p> The definition of <code class=function><dfn>completeTyping</dfn>(<span class=math>G</span>, <span class=math>Sch</span>)</code> is based on a <a>stratification</a> of <span class=math>Sch</span>. The number of <dfn>strata</dfn> of <span class=math>Sch</span> is the number of maximal strongly connected components of the <a>hierarchy and dependency graph</a> of <span class=math>Sch</span>. A <dfn>stratification</dfn> of a schema <span class=math>Sch</span> with <span class=math>k</span> strata is a function <code class=function>stratum</code> that associates with every shape label from the shape declarations of <span class=math>Sch</span> a natural number between <span class=math>1</span> and <span class=math>k</span> such that: </p> <ul> <li> If <span class=math>l1</span> and <span class=math>l2</span> belong to the same maximal strongly connected component, then <code class=function>stratum(<span class=math>l1</span>)</code> = <code class=function>stratum(<span class=math>l2</span>)</code>. </li> <li> If there is a <a>reference</a> from <span class=math>l1</span> to <span class=math>l2</span> and <span class=math>l1</span> and <span class=math>l2</span> do not belong to the same maximal strongly connected component, then <code class=function>stratum(<span class=math>s2</span>)</code> &lt; <code class=function>stratum(<span class=math>s1</span>)</code>. </li> </ul> <p> The existence of a stratification for every schema is guaranteed by the <a href=#negation-requirement>negation requirement</a>. </p> <p> Given a <a>stratification</a> <code class=function>stratum</code> of <span class=math>Sch</span> with <span class=math>k</span> strata, define inductively the series of <span class=math>k</span> typings <code class=function>completeTypingOn(<span class=math>1</span>, <span class=math>G</span>, <span class=math>Sch</span>)</code> ÔÇª <code class=function>completeTypingOn(<span class=math>k</span>, <span class=math>G</span>, <span class=math>Sch</span>)</code>. </p> <ul> <li> <code class=function>completeTypingOn(<span class=math>1</span>, <span class=math>G</span>, <span class=math>Sch</span>)</code> is the union of all correct typings that contain only RDF node/shape pairs <span class=math>(n,s)</span> with <code class=function>stratum(<span class=math>s</span>)</code> = <span class=math>1</span>; </li> <li> for every <span class=math>i</span> between <span class=math>2</span> and <span class=math>k</span>, <code class=function>completeTypingOn(<span class=math>i</span>, <span class=math>G</span>, <span class=math>Sch</span>)</code> is the union of all correct typings that: <ul> <li> contain only RDF node/shape pairs <span class=math>(n,s)</span> with <code class=function>stratum(<span class=math>s</span>)</code> Ôëñ <span class=math>i</span> </li> <li> are equal to <code class=function>completeTypingOn(<span class=math>i</span>-1, <span class=math>G</span>, <span class=math>Sch</span>)</code> when restricted to their RDF node/shape pairs <span class=math>(n1,s1)</span> for which <code class=function>stratum(<span class=math>s1</span>)</code> &lt; i. </li> </ul> </li> </ul> <p> Then <code class=function>completeTyping(<span class=math>G</span>, <span class=math>Sch</span>)</code> = <code class=function>completeTypingOn(<span class=math>k</span>, <span class=math>G</span>, <span class=math>Sch</span>)</code>. </p> <div class=note> The definition of strongly connected component and maximal strongly connected component of a graph can be found at Wikipedia: <a href=https://en.wikipedia.org/wiki/Strongly\_connected\_component>https://en.wikipedia.org/wiki/Strongly\_connected\_component</a>. </div> <div class=note> <p> The schema <span class=math>Sch</span> might have several different stratifications but <code class=function>completeTyping(<span class=math>G</span>, <span class=math>Sch</span>)</code> is the same for all these stratifications. This property is reminiscent of the use of stratified negation in Datalog. </p> <p> In order to decide <code class=function><a>isValid</a>(<span class=math>Sch</span>, <span class=math>G</span>, <span class=math>m</span>)</code>, it is sufficient to compute only a portion of the complete typing using an appropriate algorithm. </p> </div> <div class=note> <p> Popular methods for constructing the input fixed ShapeMaps can be found at <a href=https://www.w3.org/2001/sw/wiki/ShEx/ShapeMap>https://www.w3.org/2001/sw/wiki/ShEx/ShapeMap</a>. </p> </div> </section> <section id=shape-expressions> <h3>Shape Expressions</h3> <p> A <dfn data-lt="shape expressions">shape expression</dfn> is composed of four kinds of objects combined with the algebraic operators `And`, `Or` and `Not`: </p> <ul> <li> A node constraint (<span class=jobjref><a>NodeConstraint</a></span>) defines the set of allowed values of a node. These include specification of RDF node kind, literal datatype, XML string and numeric facets and enumeration of value sets. </li> <li> A shape constraint (<span class=jobjref><a>Shape</a></span>) defines a constraint on the allowed neighbourhood of a node, that is, the allowed triples that contain this node as subject or object. </li> <li> An external shape (<span class=jobjref><a>ShapeExternal</a></span>) is an extension mechanism allowing a <span class=jobjref><a>ShapeDecl</a></span> to denote an externally defined <span class=jobjref><a>shapeExpr</a></span>. <span class=note>It can be used to reference e.g. functional shapes or prohibitively large value sets.</span> </li> <li> A shape reference (<span class=jobjref><a>shapeExprLabel</a></span>) identifies another shape in the schema or an <a href=#import>imported schema</a>. </li> </ul> <section id=shape-expressions-shexj> <h4>JSON Syntax</h4> <div class=shexjTable> <table class=shexj> <tbody><tr class=nob> <th class=nob><dfn>shapeExpr</dfn></th> <td>=</td> <td><a class=objref>ShapeOr</a> | <a class=objref>ShapeAnd</a> | <a class=objref>ShapeNot</a> | <a class=objref>NodeConstraint</a> | <a class=objref>Shape</a> | <a class=objref>shapeExprRef</a> ;</td> </tr> <tr class=obj> <th class=obj><dfn>ShapeOr</dfn></th> <td>{</td> <td><dfn class=param>shapeExprs</dfn>:[<a class=nobref>shapeExpr</a>{2,}] }</td> </tr> <tr class=obj> <th class=obj><dfn>ShapeAnd</dfn></th> <td>{</td> <td><span class=param>shapeExprs</span>:[<a class=nobref>shapeExpr</a>{2,}] }</td> </tr> <tr class=obj> <th class=obj><dfn>ShapeNot</dfn></th> <td>{</td> <td><span class=param>shapeExpr</span>:<a class=nobref>shapeExpr</a> }</td> </tr> <tr class=obj> <th class=obj><dfn>ShapeExternal</dfn></th> <td>{</td> <td>}</td> </tr> <tr class=nob> <th class=nob><dfn>ShapeExprRef</dfn></th> <td>{</td> <td><span class=param>label</span>:<span class=nobref>shapeExprLabel <span class=param>exact</span>:<a>BOOL</a>? }</span></td> </tr> <tr class=nob> <th class=nob><dfn>shapeExprLabel</dfn></th> <td>=</td> <td><a>IRIREF</a> | <a>BNODE</a> ;</td> </tr> </table> </div> <div class=example> <p> Examples of shape expressions: </p> <div class="repchoice incomplete"> <pre class="json nohighlight">{ "type": "Shape", <span class=comment>ÔÇª</span> } &nbsp;</pre> <pre class="shexc nohighlight">{ <span class=comment>ÔÇª</span> } &nbsp;</pre> </div> <div style=clear:both>&nbsp;</div> <div class="repchoice incomplete"> <pre class="json nohighlight">{ "type": "ShapeAnd", "shapeExprs": [ { "type": "NodeConstraint", "nodeKind": "iri" }, { "type": "ShapeOr", "shapeExprs": [ "http://schema.example/#IssueShape", { "type": "ShapeNot", "shapeExpr": { "type": "Shape", <span class=comment>ÔÇª</span> } } ] } ] }</pre> <pre class="shexc nohighlight"> &nbsp; <span class=keyword>IRI</span> AND ( @&lt;http://schema.example/#IssueShape&gt; OR NOT { <span class=comment>ÔÇª</span> } )</pre> </div> <p> In this ShapeOr's <span class=param>shapeExprs</span>, "http://schema.example/#IssueShape" is a reference to the <a>shape expression</a> with the id "http://schema.example/#IssueShape". </p> </div> </section> <section id=shape-expression-semantics> <h4>Semantics</h4> <p> For a node n of the graph G, we define <dfn>neigh</dfn>(n) as the set of triples adjacent to n in the graph; these are the triples that have n either as subject or as object. </p> <p> For a shape expression <span class=var>se</span> we define its set of shapes <dfn>nestedShapes</dfn><span class=math>(<span class=param>se</span>)</span> recursively on the structure of <span class=var>se</span>: </p> <ul> <li>if <span class=var>se</span> is a <span class=jobjref><a class=obj>NodeConstraint</a></span>, then <span class=math><a>nestedShapes</a>(<span class=var>se</span>) = emptyset</span></li> <li>if <span class=var>se</span> is a <span class=jobjref><a class=obj>Shape</a></span>, then <span class=math><a>nestedShapes</a>(<span class=var>se</span>) = {se}</span></li> <li>if <span class=var>se</span> is a <span class=jobjref><a class=obj>ShapeNot</a></span>, then <span class=math><a>nestedShapes</a>(<span class=var>se</span>) = shapes(se.shapeExpr)</span></li> <li>if <span class=var>se</span> is a <span class=jobjref><a class=obj>ShapeAnd</a></span> or <span class=jobjref><a class=obj>ShapeOr</a></span>, then <span class=math><a>nestedShapes</a>(<span class=var>se</span>)</span> is the union of the sets <span class=math><a>nestedShapes</a>(<span class=var>se2</span>)</span> for all <span class=var>se2</span> in <span class=param>se.shapeExprs</span></li> <li>if <span class=var>se</span> is a <span class=jobjref><a class=obj>shapeExprRef</a></span> with label <span class=var>lab</span>, then <span class=math><a>nestedShapes</a>(<span class=var>se</span>) = <a>nestedShapes</a>(<a>def</a>(<span class=var>L</span>))</span></li> <li>if <span class=var>se</span> is a <span class=jobjref><a class=obj>ShapeExternal</a></span>, then <span class=math><a>nestedShapes</a>(<span class=var>se</span>)</span> is the set of shapes denoted by <span class=var>se</span>. </li> </ul> <p> For shape expression labels <span class=var>label1</span>, <span class=var>label2</span>, we say that <span class=var>label2</span> <dfn>directly extends</dfn> <span class=var>label1</span> if <span class=math><a>nestedShapes</a>(<a>def</a>(<span class=var>label2</span>))<span class=math> contains a Shape <span class=var>s</span> such that <span class=param>s.extends</span> contains <span class=var>label2</span>. The <dfn>extension hierarchy graph</dfn> of a shapes schema is a directed graph whose nodes are the shape expression labels of the schema and that has an edge from <span class=var>label2</span> to <span class=var>label1</span> whenever <span class=var>label2</span> <a>directly extends</a> <span class=var>label1</span>. </span></span></p> <p> <dfn>satisfies</dfn>: The expression <code class=function>satisfies(<span class=math>n</span>, <span class=math>se</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code> indicates that a node <span class=math>n</span>, a subset <span class=math>R</span> of <span class=math>neigh(n)</span>, and a <a>graph</a> <span class=math>G</span> satisfy a <a>shape expression</a> <span class=math>se</span> with <a>typing</a> <span class=math>t</span> for schema <span class=math>Sch</span>.<br> <dfn>satisfiesDescendant</dfn>: The expression <code class=function>satisfiesDescendant(<span class=math>n</span>, <span class=var>shapeExprLabel</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code> indicates that <span class=math>n</span>, a subset <span class=math>R</span> of <span class=math>neigh(n)</span>, and <span class=math>G</span> and some non-abstract <span class=var>child</span> of <span class=var>shapeExprLabel</span> in the <a>extension hierarchy graph</a> <code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=var>child</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code>, with the given <a>typing</a> <span class=math>t</span>. </p> <p> <code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=math>se</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code> is true if and only if: </p> <ul> <li id=satisfies-NodeConstraint><span class=math>se</span> is a <span class=jobjref><a class=obj>NodeConstraint</a></span> and <code class=function><a href=#satisfies2-NodeConstraint>satisfies2</a>(<span class=math>n</span>, <span class=math>se</span>)</code> as described below in <a href=#node-constraints>Node Constraints</a>. Note that testing if a node satisfies a node constraint does not require a <a>graph</a> or <a>typing</a>.</li> <li id=satisfies-Shape-proxy><span class=math>se</span> is a <span class=jobjref><a class=obj>Shape</a></span> and <code class=function><a>matchesShape</a>(<span class=math>n</span>, <span class=math>S</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>, <span class=math>R</span>)</code> is true. </li> <li id=satisfies-ShapeOr><span class=math>se</span> is a <span class=jobjref><a class=obj>ShapeOr</a></span> and there is some <a>shape expression</a> <span class=math>se2</span> in <span class=math>se</span>.<span class=param>shapeExprs</span> such that <code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=math>se2</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code>. </li> <li id=satisfies-ShapeAnd><span class=math>se</span> is a <span class=jobjref><a class=obj>ShapeAnd</a></span> and for every <a>shape expression</a> <span class=math>se2</span> in <span class=math>se</span>.<span class=param>shapeExprs</span>, <code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=math>se2</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code>. </li> <li id=satisfies-ShapeNot><span class=math>se</span> is a <span class=jobjref><a class=obj>ShapeNot</a></span> and for the <a>shape expression</a> <span class=math>se2</span> at <span class=math>se</span>.<span class=param>shapeExpr</span>, <code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=math>se2</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code> is false.</li> <li id=satisfies-ShapeExternal><span class=math>se</span> is a <span class=jobjref><a class=obj>ShapeExternal</a></span> and implementation-specific mechansims not defined in this specification indicate success.</li> <li id=satisfies-shapeExprRef><span class=math>se</span> is a <span class=jobjref><a class=obj>ShapeExprRef</a></span>. If <span class=param>ShapeExprRef.exact</span>, <code class=function><a>satisfies</a>(<span class=math>n</span>, <a>def</a>(<span class=math>se.label</span>), <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code>, otherwise <code class=function><a>satisfiesDescendant</a>(<span class=math>n</span>, <span class=math>se.label</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>t</span>, <span class=math>R</span>)</code>. </li> </ul> <div class=example> <p> Given the three shape expressions <span class=math>SE<sub>1</sub></span>, <span class=math>SE<sub>2</sub></span>, <span class=math>SE<sub>3</sub></span> in a <span class=jobjref><a class=obj>Schema</a></span> <span class=math>Sch</span>, such that: </p> <ul> <li><code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=math>SE<sub>1</sub></span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>)</code> </li> <li><code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=math>SE<sub>2</sub></span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>)</code> </li> <li>NOT <code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=math>SE<sub>3</sub></span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>)</code> </li> </ul> <p> the following hold: </p> <ul class=codelist> <li> <div> <div style=float:left class="code function">satisfies(</div> <div style=float:left class=code><span class=math>n</span>,<br> <div class="inline json"> <pre>{ "type": "ShapeAnd", "shapeExprs": [ <span class=comment>SE<sub>1</sub></span>, <span class=comment>SE<sub>2</sub></span> ] }</pre> </div>,<br> <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>) </div> <div style=clear:both></div> </div> </li> <li> <div> <div style=float:left class="code function">satisfies(</div> <div style=float:left class=code><span class=math>n</span>,<br> <div class="inline json"> <pre>{ "type": "ShapeOr", "shapeExprs": [ <span class=comment>SE<sub>1</sub></span>, <span class=comment>SE<sub>2</sub></span>, <span class=comment>SE<sub>3</sub></span> ] }</pre> </div>,<br> <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>) </div> <div style=clear:both></div> </div> </li> <li> <div>NOT &nbsp;<div style=float:left class="code function">satisfies(</div> <div style=float:left class=code><span class=math>n</span>,<br> <div class="inline json code"> <pre>{ "type": "ShapeNot", "shapeExpr": { { "type": "ShapeOr", "shapeExprs": [ <span class=comment>SE<sub>1</sub></span>, { "type": "ShapeAnd", "shapeExprs": [ <span class=comment>SE<sub>2</sub></span>, <span class=comment>SE<sub>3</sub></span> ] } ] } } }</pre> </div>,<br> <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>) </div> <div style=clear:both></div> </div> </li> </ul> <p> If <span class=math>Sch</span>'s <a class=param>shapes</a> maps "<code>http://schema.example/#shape1</code>" to <span class=math>SE<sub>1</sub></span> then the following holds: </p> <ul class=codelist> <li> <div> <div style=float:left class="code function">satisfies(</div> <div style=float:left class=code><span class=math>n</span>,<br> <div class="inline json"> <pre>http://schema.example/#shape1"</pre> </div>,<br> <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>) </div> <div style=clear:both></div> </div> </li> </ul> </div> <div class=example> <p> In this example, EmployeeShape <a>directly extends</a> PersonShape and transitively extends EntityShape </p> <div class=repchoice> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id" : "http://schema.example/#EntityShape", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#entityId" } }, { "id" : "http://schema.example/#PersonShape", "type" : "Shape", "extends" : [ "http://schema.example/#EntityShape" ], "expression" : { "type" : "TripleConstraint", "predicate" : "http://xmlns.com/foaf/0.1/name" } }, { "id" : "http://schema.example/#EmployeeShape", "type" : "Shape", "extends" : [ "http://schema.example/#PersonShape" ], "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#employeeNumber" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:EntityShape { ex:entityId . } ex:PersonShape EXTENDS @ex:EntityShape { foaf:name . } ex:EmployeeShape EXTENDS @ex:PersonShape { ex:employeeNumber . } &nbsp; </pre> </div> </div> <div class=example> <p> In this example, UserShape <a>directly extends</a> PersonShape, and PersonShape directly references a conjunct which <a>directly extends</a> EntityShape. Through this, UserShape transitively extends EntityShape. </p> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#EntityShape", "type": "Shape", "closed": true, "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#entityId" } }, { "id": "http://schema.example/#PersonShape", "type": "ShapeAnd", "shapeExprs": [ { "type": "Shape", "extends": [ "http://schema.example/#EntityShape" ], "closed": true, "expression": { "type": "TripleConstraint", "predicate": "http://xmlns.com/foaf/0.1/name" } }, { "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#entityId", "valueExpr": { "type": "NodeConstraint", "datatype": "http://www.w3.org/2001/XMLSchema#integer" } } } ] }, { "id": "http://schema.example/#UserShape", "type": "Shape", "extends": [ "http://schema.example/#PersonShape" ], "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#userId" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; &lt;#EntityShape&gt; CLOSED { :entityId . } &lt;#PersonShape&gt; EXTENDS @&lt;#EntityShape&gt; CLOSED { foaf:name . } AND { :entityId xsd:integer } &lt;#UserShape&gt; EXTENDS @&lt;PersonShape&gt;{ :userId . } &nbsp; </pre> </div> </div> </section> </section> <section id=node-constraints> <h3>Node Constraints</h3> <div class=shexjTable> <table class=shexj> <tbody><tr class=obj> <th class=obj><dfn>NodeConstraint</dfn></th> <td>{</td> <td><span class=param>nodeKind</span>:(<span class=literal>"iri"</span> | <span class=literal>"bnode"</span> | <span class=literal>"nonliteral"</span> | <span class=literal>"literal"</span>)? <span class=param>datatype</span>:<a>IRIREF</a>? <a class=nobref>xsFacet</a>\* <span class=param>values</span>:[<a class=nobref>valueSetValue</a>+]? }</td> </tr> <tr class=nob> <th class=nob><dfn>xsFacet</dfn></th> <td>=</td> <td><a class=nobref>stringFacet</a> | <a class=nobref>numericFacet</a> ;</td> </tr> <tr class=nob> <th class=nob><dfn>stringFacet</dfn></th> <td>=</td> <td>(<span class=param>length</span>|<span class=param>minlength</span>|<span class=param>maxlength</span>):<a>INTEGER</a> | <span class=param>pattern</span>:<a>STRING</a> <span class=param>flags</span>:<a>STRING</a>? ;</td> </tr> <tr class=nob> <th class=nob><dfn>numericFacet</dfn></th> <td>=</td> <td>(<span class=param>mininclusive</span>|<span class=param>minexclusive</span>|<span class=param>maxinclusive</span>|<span class=param>maxexclusive</span>):<a class=nobref>numericLiteral</a> </td> </tr> <tr> <th></th> <td>|</td> <td>(<span class=param>totaldigits</span>|<span class=param>fractiondigits</span>):<a>INTEGER</a> ;</td> </tr> <tr class=nob> <th class=nob><dfn>numericLiteral</dfn></th> <td>=</td> <td><a>INTEGER</a> | <a>DECIMAL</a> | <a>DOUBLE</a> ;</td> </tr> <tr class=nob> <th class=nob><dfn>valueSetValue</dfn></th> <td>=</td> <td><a class=nobref>objectValue</a> | <a class=objref>IriStem</a> | <a class=objref>IriStemRange</a> | <a class=objref>LiteralStem</a> | <a class=objref>LiteralStemRange</a> | <a class=objref>Language</a> | <a class=objref>LanguageStem</a> | <a class=objref>LanguageStemRange</a> ;</td> </tr> <tr class=nob> <th class=nob><dfn>objectValue</dfn></th> <td>=</td> <td><a>IRIREF</a> | <a class=objref>ObjectLiteral</a> ;</td> </tr> <tr class=nob> <th class=nob><dfn>ObjectLiteral</dfn></th> <td>{</td> <td><span class=param>value</span>:<a>STRING</a> <span class=param>language</span>:<a>STRING</a>? <span class=param>type</span>:<a>STRING</a>? }</td> </tr> <tr class=obj> <th class=obj><dfn>IriStem</dfn></th> <td>{</td> <td><span class=param>stem</span>:<a>IRIREF</a> }</td> </tr> <tr class=obj> <th class=obj><dfn>IriStemRange</dfn></th> <td>{</td> <td><span class=param>stem</span>:(<a>IRIREF</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>IRIREF</a>|<a class=objref>IriStem</a>+]? }</td> </tr> <tr class=obj> <th class=obj><dfn>LiteralStem</dfn></th> <td>{</td> <td><span class=param>stem</span>:<a>STRING</a> }</td> </tr> <tr class=obj> <th class=obj><dfn>LiteralStemRange</dfn></th> <td>{</td> <td><span class=param>stem</span>:(<a class=nobref>STRING</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>STRING</a>|<a class=objref>LiteralStem</a>+]? }</td> </tr> <tr class=obj> <th class=obj><dfn>Language</dfn></th> <td>{</td> <td><span class=param>languageTag</span>:<a>LANGTAG</a> }</td> </tr> <tr class=obj> <th class=obj><dfn>LanguageStem</dfn></th> <td>{</td> <td><span class=param>stem</span>:<a>LANGTAG</a> }</td> </tr> <tr class=obj> <th class=obj><dfn>LanguageStemRange</dfn></th> <td>{</td> <td><span class=param>stem</span>:(<a>LANGTAG</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>LANGTAG</a>|<a class=objref>LanguageStem</a>+]? }</td> </tr> <tr class=obj> <th class=obj><dfn>Wildcard</dfn></th> <td>{</td> <td><span class=comment>/\* empty \*/</span> }</td> </tr> </table> </div> <section id=node-constraint-semantics> <h4>Semantics</h4> <p id=satisfies2-NodeConstraint> For a node <span class=param>n</span> and constraint <span class=param>nc</span>, <code class=function>satisfies2(<span class=math>n</span>, <span class=math>nc</span>)</code> if and only if for every <span class=param>nodeKind</span>, <span class=param>datatype</span>, <span class=param>xsFacet</span> and <span class=param>values</span> constraint value <span class=math>v</span> present in <span class=param>nc</span> <code class=function>nodeSatisfies(<span class=math>n</span>, <span class=math>v</span>)</code>. The following sections define <code class=function>nodeSatisfies</code> for each of these types of constraints: </p> <ul> <li><a href=#nodeKind>Node Kind Constraints</a></li> <li><a href=#datatype>Datatype Constraints</a></li> <li><a href=#xs-string>XML Schema String Facet Constraints</a></li> <li><a href=#xs-numeric>XML Schema Numeric Facet Constraints</a></li> <li><a href=#values>Values Constraints</a></li> </ul> </section> <section id=nodeKind> <h4>Node Kind Constraints</h4> <p id=nodeSatisfies-nodeKind> For a node <span class=math>n</span> and constraint value <span class=math>v</span>, <code class=function>nodeSatisfies(<span class=math>n</span>, <span class=math>v</span>)</code> if: </p> <ul> <li><span class=math>v</span> = "iri" and <span class=math>n</span> is an <a>IRI</a>.</li> <li><span class=math>v</span> = "bnode" and <span class=math>n</span> is a <a>blank node</a>. </li> <li><span class=math>v</span> = "literal" and <span class=math>n</span> is a <a>Literal</a>. </li> <li><span class=math>v</span> = "nonliteral" and <span class=math>n</span> is an <a>IRI</a> or <a>blank node</a>.</li> </ul> <div class=example> <div class="example-title marker"><span>Node Kind example 1</span></div> <p> The following examples use a <span class=jobjref><a>TripleConstraint</a></span> object described later in the document. The </p> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#state", "valueExpr": { "type": "NodeConstraint", "nodeKind": "iri" } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:state IRI } &nbsp; </pre> </div> <pre class=data>&lt;issue1&gt; ex:state ex:HunkyDory . &lt;issue2&gt; <span style="border-bottom:thin solid red">ex:taste</span> ex:GoodEnough . <span class=fail>&lt;issue3&gt; ex:state "just fine" .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue1&gt;</td> <td>&lt;IssueShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue2&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail>expected 1 <code>ex:state</code> property.</td> </tr> <tr class=fail> <td>&lt;issue3&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail><code>ex:state</code> expected to be an IRI, literal found.</td> </tr> </table> <p> Note that &lt;issue2&gt; <span class=fail>fails</span> not because of a nodeKind violation but instead because of a <a href=#matches-cardinality>Cardinality</a> violation described below. </p> </div> </section> <section id=datatype> <h4>Datatype Constraints</h4> <p id=nodeSatisfies-datatype> For a node <span class=math>n</span> and constraint value <span class=math>v</span>, <code class=function>nodeSatisfies(<span class=math>n</span>, <span class=math>v</span>)</code> if <span class=math>n</span> is a Literal with the datatype <span class=math>v</span> and, if <span class=math>v</span> is in the set of <a data-cite=sparql11-query#operandDataTypes>SPARQL operand data types</a>[[!sparql11-query]], an XML schema string with a value of the lexical form of <span class=math>n</span> can be cast to the target type <span class=math>v</span> per <a data-cite=xpath-functions-31#casting>XPath Functions 3.1 section 19 Casting</a>[[!xpath-functions]]. The lexical form and numeric value (where applicable) of all datatypes required by <a data-cite=sparql11-query#FunctionMapping>SPARQL XPath Constructor Functions</a> MUST be tested for conformance with the corresponding XML Schema form. ShEx extensions MAY add support for other datatypes. </p> <div class=example> <div class="example-title marker"><span>Datatype example 1</span></div> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#submittedOn", "valueExpr": { "type": "NodeConstraint", "datatype": "http://www.w3.org/2001/XMLSchema#date" } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:submittedOn xsd:date } &nbsp; </pre> </div> <pre class=data>&lt;issue1&gt; ex:submittedOn "2016-07-08"^^xsd:date . <span class=fail>&lt;issue2&gt; ex:submittedOn "2016-07-08T01:23:45Z"^^xsd:dateTime .</span> <span class=fail>&lt;issue3&gt; ex:submittedOn "2016-07"^^xsd:date .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue1&gt;</td> <td>&lt;IssueShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue2&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail><code>ex:submittedOn</code> expected to be an <code>xsd:date</code>, <code>xsd:dateTime</code> found.</td> </tr> <tr class=fail> <td>&lt;issue3&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail><code>2016-07</code> is not a valid <code>xsd:date</code>.</td> </tr> </table> </div> <div class=note> <p> In RDF 1.1, <dfn data-cite=rdf11-concepts#dfn-language-tagged-string data-lt="language-tagged string">language-tagged strings</dfn>[[!rdf11-concepts]] have the datatype <code>http://www.w3.org/1999/02/22-rdf-syntax-ns#langString</code>. </p> <p> RDF 1.0 included <a>RDF literals</a> with no datatype or language tag. These are called "<a data-cite=sparql11-query#simple\_literal>simple literals</a>" in SPARQL11[[!sparql11-query]]. In RDF 1.1, these literals have the datatype <code>http://www.w3.org/2001/XMLSchema#string</code>. </p> </div> <div class=example> <div class="example-title marker"><span>Datatype example 2</span></div> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://www.w3.org/2000/01/rdf-schema#label", "valueExpr": { "type": "NodeConstraint", "datatype": "http://www.w3.org/1999/02/22-rdf-syntax-ns#langString" } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { rdfs:label rdf:langString } &nbsp; </pre> </div> <pre class=data>&lt;issue3&gt; rdfs:label "emits dense black smoke"@en . <span class=fail>&lt;issue4&gt; rdfs:label "unexpected odor" .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue3&gt;</td> <td>&lt;IssueShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue4&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail><code>rdfs:label</code> expected to be an <code>rdf:langString</code>, <code>xsd:string</code> found.</td> </tr> </table> </div> </section> <section id=xs-string> <h4>XML Schema String Facet Constraints</h4> <p> String facet constraints apply to the lexical form of the <a>RDF Literals</a> and <a>IRIs</a> and <a>blank node identifiers</a> (see <a href=#blank-node-label>note below</a> regarding access to <a>blank node identifiers</a>).<br> Let <span class=math>lex</span> = </p> <ul> <li>if the value <span class=math>n</span> is an <a>RDF Literal</a>, the <a data-cite=rdf11-concepts#dfn-lexical-form>lexical form</a> of the literal (see [[!rdf11-concepts]] <a data-cite=rdf11-concepts#section-Graph-Literal>section 3.3 Literals</a>).</li> <li>if the value <span class=math>n</span> is an <a>IRI</a>, the <a data-cite=rdf11-concepts#dfn-iri>IRI string</a> (see [[!rdf11-concepts]] <a data-cite=rdf11-concepts#section-iris>section 3.2 IRIs</a>).</li> <li>if the value <span class=math>n</span> is a <a>blank node</a>, the <a>blank node identifier</a> (see [[!rdf11-concepts]] <a data-cite=rdf11-concepts#section-blank-nodes>section 3.4 Blank Nodes</a>).</li> </ul> <p> Let <span class=math>len</span> = the number of unicode codepoints in <span class=math>lex</span><br> For a node <span class=math>n</span> and constraint value <span class=math>v</span>, <code class=function>nodeSatisfies(<span class=math>n</span>, <span class=math>v</span>)</code>: </p> <ul> <li id=nodeSatisfies-length>for "<code>length</code>" constraints, <span class=math>v = len</span>,</li> <li id=nodeSatisfies-minlength>for "<code>minlength</code>" constraints, <span class=math>v &gt;= len</span>,</li> <li id=nodeSatisfies-maxlength>for "<code>maxlength</code>" constraints, <span class=math>v &lt;= len</span>,</li> <li id=nodeSatisfies-pattern> for "<code>pattern</code>" constraints, <span class=math>v</span> is unescaped into a valid <a data-cite=xpath-functions-31#regex-syntax>XPath 3.1 regular expression</a>[[!xpath-functions-31]] <span class=math>re</span> and invoking <code class=function><a data-cite=xpath-functions-31#func-matches>fn:matches</a>(<span class=math>lex</span>, <span class=math>re</span>)</code> returns <code>fn:true</code>. If the <span class=param>flags</span> parameter is present, it is passed as a third argument to <code class=function>fn:matches</code>. The pattern may have XPath 3.1 regular expression escape sequences per the modified production [10] in <a href=https://www.w3.org/TR/xpath-functions-31/#matching-start-and-end>section 5.6.1.1</a> as well as numeric escape sequences of the form 'u' HEX HEX HEX HEX or 'U' HEX HEX HEX HEX HEX HEX HEX HEX. Unescaping replaces numeric escape sequences with the corresponding unicode codepoint. </li> </ul> <div class=example> <div class="example-title marker"><span>String Facets example 1</span></div> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#submittedBy", "valueExpr": { "type": "NodeConstraint", "minlength": 10 } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:submittedBy MINLENGTH 10 } &nbsp; </pre> </div> <pre class=data>&lt;issue1&gt; ex:submittedBy &lt;http://a.example/bob&gt; . <span class=comment># 20 characters</span> <span class=fail>&lt;issue2&gt; ex:submittedBy "Bob" .</span> <span class=comment># 3 characters</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue1&gt;</td> <td>&lt;IssueShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue2&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail><code>ex:submittedBy</code> expected to be &gt;= 10 characters,<br> 3 characters found.</td> </tr> </table> </div> <div class=note id=blank-node-label> <p> Access to <a>blank node identifiers</a> may be impossible or unadvisable for many use cases. For instance, the SPARQL Query and SPARQL Update languages treat blank nodes in the query, labeled or otherwise, as variables. Lexical constraints on <a>blank node identifiers</a> can only be implemented in systems which preserve such labels on data import. </p> </div> <div class=example> <div class="example-title marker"><span>String Facets example 2</span></div> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#submittedBy", "valueExpr": { "type": "NodeConstraint", "pattern": "genuser[0-9]+", "flags": "i" } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:submittedBy /genuser[0-9]+/i } &nbsp; </pre> </div> <pre class=data>&lt;issue6&gt; ex:submittedBy \_:genUser218 . <span class=fail>&lt;issue7&gt; ex:submittedBy \_:genContact817 .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue6&gt;</td> <td>&lt;IssueShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue7&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail><code>\_:genContact817</code> expected to match <code>genuser[0-9]+</code>. </td> </tr> </table> </div> <p> When expressed as JSON strings, regular expressions are subject to the JSON string escaping rules. </p> <div class=example> <div class="example-title marker"><span>String Facets example 3</span></div> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#ProductShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#trademark", "valueExpr": { "type": "NodeConstraint", "pattern": "^/\\t\\\\\uD835\uDCB8\\?$" } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:ProductShape { ex:trademark /^\/\t\\\U0001D4B8\?$/ } &nbsp; </pre> </div> <pre class=data>&lt;product6&gt; ex:trademark " \\­ØÆ©?" . &lt;product7&gt; ex:trademark "\t\\\U0001D4B8?" . <span class=comment># Turtle literals have escape characters <a href=https://www.w3.org/TR/turtle/#grammar-production-ECHAR>[tbnrf"'\]</a>.</span> <span class=fail>&lt;product8&gt; ex:trademark "\t\\\\U0001D4B8?" .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;product6&gt;</td> <td>&lt;ProductShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=pass> <td>&lt;product7&gt;</td> <td>&lt;ProductShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;product8&gt;</td> <td>&lt;ProductShape&gt;</td> <td>fail</td> <td class=fail>found "<code>\U0001D4B8</code>" instead of "<code>­ØÆ©</code>" (codepoint U+1D4B8).</td> </tr> </table> </div> </section> <section id=xs-numeric> <h4>XML Schema Numeric Facet Constraints</h4> <p> Numeric facet constraints apply to the numeric value of <a>RDF Literals</a> with datatypes listed in <a data-cite=sparql11-query#operandDataTypes>SPARQL 1.1 Operand Data Types</a>[[!sparql11-query]]. Numeric constraints on non-numeric values fail. <code>totaldigits</code> and <code>fractiondigits</code> constraints on values not derived from <code>xsd:decimal</code> fail. </p> <p> Let <span class=math>num</span> be the numeric value of <span class=math>n</span>.<br> For a node <span class=math>n</span> and constraint value <span class=math>v</span>, <code class=function>nodeSatisfies(<span class=math>n</span>, <span class=math>v</span>)</code>: </p> <ul> <li id=nodeSatisfies-mininclusive>for "<code>mininclusive</code>" constraints, <span class=math>v &lt;= num</span>,</li> <li id=nodeSatisfies-minexclusive>for "<code>minexclusive</code>" constraints, <span class=math>v &lt; num</span>,</li> <li id=nodeSatisfies-maxinclusive>for "<code>maxinclusive</code>" constraints, <span class=math>v &gt;= num</span>,</li> <li id=nodeSatisfies-maxexclusive>for "<code>maxexclusive</code>" constraints, <span class=math>v &gt; num</span>,</li> <li id=nodeSatisfies-totaldigits>for "<code>totaldigits</code>" constraints, <span class=math>v</span> is less than or equals the number of digits in the <a data-cite=xmlschema-2#dt-canonical-representation>XML Schema canonical form</a>[[!xmlschema-2]] of the value of <span class=math>n</span>,</li> <li id=nodeSatisfies-fractiondigits>for "<code>fractiondigits</code>" constraints, <span class=math>v</span> is less than or equals the number of digits to the right of the decimal place in the <a data-cite=xmlschema-2#dt-canonical-representation>XML Schema canonical form</a>[[!xmlschema-2]] of the value of <span class=math>n</span>, ignoring trailing zeros.</li> </ul> <p> The operators <span class=math>&lt;=</span>, <span class=math>&lt;</span>, <span class=math>&gt;=</span> and <span class=math>&gt;</span> are evaluated after performing <a data-cite=xpath20#promotion>numeric type promotion</a>[[!xpath20]]. </p> <div class=example> <div class="example-title marker"><span>Numeric Facets example 1</span></div> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#confirmations", "valueExpr": { "type": "NodeConstraint", "mininclusive": 1 } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:confirmations MININCLUSIVE 1 } &nbsp; </pre> </div> <pre class=data>&lt;issue1&gt; ex:confirmations 1 . &lt;issue2&gt; ex:confirmations 2^^xsd:byte . <span class=fail>&lt;issue3&gt; ex:confirmations 0 .</span> <span class=fail>&lt;issue4&gt; ex:confirmations "ii"^^ex:romanNumeral .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue1&gt;</td> <td>&lt;IssueShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=pass> <td>&lt;issue2&gt;</td> <td>&lt;IssueShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue3&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail><code>0</code> is less than <code>1</code>.</td> </tr> <tr class=fail> <td>&lt;issue4&gt;</td> <td>&lt;IssueShape&gt;</td> <td>fail</td> <td class=fail><code>ex:romanNumeral</code> is not a numeric datatype.</td> </tr> </table> </div> </section> <section id=values> <h4>Values Constraint</h4> <p> The <code>nodeSatisfies</code> semantics for <span class=jobjref><a class=obj>NodeConstraint</a></span> <span class=param>values</span> depends on a <code>nodeIn</code> function <a href=#nodeIn>defined below</a>. </p> <p> For a node <span class=math>n</span> and constraint value <span class=math>v</span>, <code class=function>nodeSatisfies(<span class=math>n</span>, <span class=math>v</span>)</code> if <span class=math>n</span> matches some <a class=nobref>valueSetValue</a> <span class=math>vsv</span> in <span class=math>v</span>. A term matches a valueSetValue if: </p> <ul> <li><span class=math id=nodeSatisfies-objectValue>vsv</span> is an <a class=nobref>objectValue</a> and <span class=math>n = vsv</span>.</li> <li><span class=math>vsv</span> is a <span class=jobjref><a class=obj>Language</a></span> with <span class=param>languageTag</span> <span class=math>lt</span> and <span class=math>n</span> is a <a>language-tagged string</a> with a <a>language tag</a> <span class=math>l</span> and <span class="math function">l = lt</span>.</li> <li><span class=math id=nodeSatisfies-Stem>vsv</span> is a <span class=jobjref><a class=nobref>IriStem</a></span>, <span class=jobjref><a class=nobref>LiteralStem</a></span> or <span class=jobjref><a class=nobref>LanguageStem</a></span> with <span class=param>stem</span> <span class=math>st</span> and <span class=math>nodeIn(n, st)</span>.</li> <li><span class=math id=nodeSatisfies-StemRange>vsv</span> is a <span class=jobjref><a class=nobref>IriStemRange</a></span>, <span class=jobjref><a class=nobref>LiteralStemRange</a></span> or <span class=jobjref><a class=nobref>LanguageStemRange</a></span> with <span class=param>stem</span> <span class=math>st</span> and <span class=param>exclusions</span> <span class=math>excls</span> and <span class="math function">nodeIn(n, st)</span> and there is no <span class=math>x</span> in <span class=math>excls</span> such that <span class="math function">nodeIn(n, excl)</span>.</li> <li><span class=math id=nodeSatisfies-Wildcard>vsv</span> is a <span class=jobjref><a class=nobref>Wildcard</a></span> with <span class=param>exclusions</span> <span class=math>excls</span> and there is no <span class=math>x</span> in <span class=math>excls</span> such that <span class="math function">nodeIn(n, excl)</span>.</li> </ul> <p> <span class=label id=nodeIn>nodeIn</span>: asserts that an RDF node <span class=math>n</span> is equal to an RDF term <span class=math>s</span> or is in a set defined by a <span class=jobjref><a class=obj>IriStem</a></span>, <span class=jobjref><a class=obj>LiteralStem</a></span> or <span class=jobjref><a class=obj>LanguageStem</a></span>.<br> The expression <code class=function>nodeIn(<span class=math>n</span>, <span class=math>s</span>)</code> is satisfied if: </p> <ul> <li><span class=math>n = s</span>.</li> <li><span class=math>s</span> is a <span class=jobjref><a class=obj>IriStem</a></span>, <span class=jobjref><a class=obj>LiteralStem</a></span> or <span class=jobjref><a class=obj>LanguageStem</a></span> with <span class=param>stem</span> <span class=math>st</span> and: <ul> <li><span class=math>s</span> is a <span class=jobjref><a class=obj>IriStem</a></span> and <span class=math>n</span> is an <a>IRI</a> and <span class="math function"><a data-cite=xpath-functions-31#func-starts-with>fn:starts-with</a>(n, st)</span>.</li> <li><span class=math>s</span> is a <span class=jobjref><a class=obj>LiteralStem</a></span> and <span class=math>n</span> is an <a>RDF Literal</a> with a lexical value <span class=math>l</span> and <span class="math function"><a data-cite=xpath-functions-31#func-starts-with>fn:starts-with</a>(l, st)</span>.</li> <li> <span class=math>s</span> is a <span class=jobjref><a class=obj>LanguageStem</a></span>, <span class=math>n</span> is a <a>language-tagged string</a> with a <a>language tag</a> <span class=math>l</span>, <span class=math>st</span> is a basic language range per <a class=norm href=http://www.ietf.org/rfc/rfc4647.txt>Matching of Language Tags</a> [[!rfc4647]] section 2.1 and <span class=math>l</span> matches <span class=math>st</span> per the basic filtering scheme defined in [[!rfc4647]] section 3.3.1. The basic language range wildcard ("\*") is represented by an empty <span class=param>stem</span> (""). </li> </ul> </li> </ul> <div class=example> <div class="example-title marker"><span>Values Constraint example 1</span></div> <p>NoActionIssueShape requires a state of Resolved or Rejected:</p> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#NoActionIssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#state", "valueExpr": { "type": "NodeConstraint", "values": [ "http://schema.example/#Resolved", "http://schema.example/#Rejected" ] } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:NoActionIssueShape { ex:state [ ex:Resolved ex:Rejected ] } &nbsp; </pre> </div> <pre class=data>&lt;issue1&gt; ex:state ex:Resolved . <span class=fail>&lt;issue2&gt; ex:state ex:Unresolved .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue1&gt;</td> <td>&lt;NoActionIssueShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue2&gt;</td> <td>&lt;NoActionIssueShape&gt;</td> <td>fail</td> <td class=fail><code>ex:state</code> expected to be <code>ex:Resolved</code> or <code>ex:Rejected</code>, <code>ex:Unresolved</code> found.</td> </tr> </table> </div> <div class=example> <div class="example-title marker"><span>Values Constraint example 2</span></div> <p>An employee must have an email address that is the string "N/A" or starts with "engineering-" or "sales-" but not "sales-contacts" or "sales-interns":</p> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#EmployeeShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://xmlns.com/foaf/0.1/mbox", "valueExpr": { "type": "NodeConstraint", "values": [ {"value": "N/A"}, { "type": "IriStem", "stem": "mailto:engineering-" }, { "type": "IriStemRange", "stem": "mailto:sales-", "exclusions": [ { "type": "IriStem", "stem": "mailto:sales-contacts" }, { "type": "IriStem", "stem": "mailto:sales-interns" } ] } ] } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:EmployeeShape { foaf:mbox [ "N/A" &lt;mailto:engineering-&gt;~ &lt;mailto:sales-&gt;~ - &lt;mailto:sales-contacts&gt;~ - &lt;mailto:sales-interns&gt;~ ] } &nbsp; </pre> </div> <pre class=data>&lt;issue3&gt; foaf:mbox "N/A" . &lt;issue4&gt; foaf:mbox &lt;mailto:engineering-2112@a.example&gt; . &lt;issue5&gt; foaf:mbox &lt;mailto:sales-835@a.example&gt; . <span class=fail>&lt;issue6&gt; foaf:mbox "missing" .</span> <span class=fail>&lt;issue7&gt; foaf:mbox &lt;mailto:sales-contacts-999@a.example&gt; .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue3&gt;</td> <td>&lt;EmployeeShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=pass> <td>&lt;issue4&gt;</td> <td>&lt;EmployeeShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=pass> <td>&lt;issue5&gt;</td> <td>&lt;EmployeeShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue6&gt;</td> <td>&lt;EmployeeShape&gt;</td> <td>fail</td> <td class=fail><code>"missing"</code> is not in value set.</td> </tr> <tr class=fail> <td>&lt;issue7&gt;</td> <td>&lt;EmployeeShape&gt;</td> <td>fail</td> <td class=fail><code>&lt;mailto:sales-contacts-999@a.example&gt;</code> is excluded.</td> </tr> </table> </div> <div class=example> <div class="example-title marker"><span>Values Constraint example 3</span></div> <p>An employee must not have an email address that starts with "engineering-" or "sales-":</p> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#EmployeeShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://xmlns.com/foaf/0.1/mbox", "valueExpr": { "type": "NodeConstraint", "values": [ { "type": "IriStemRange", "stem": {"type": "Wildcard"}, "exclusions": [ { "type": "IriStem", "stem": "mailto:engineering-" }, { "type": "IriStem", "stem": "mailto:sales-" } ] } ] } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:EmployeeShape { foaf:mbox [ . - &lt;mailto:engineering-&gt;~ - &lt;mailto:sales-&gt;~ ] } &nbsp; </pre> </div> <pre class=data>&lt;issue8&gt; foaf:mbox 123 . &lt;issue9&gt; foaf:mbox &lt;mailto:core-engineering-2112@a.example&gt; . <span class=fail>&lt;issue10&gt; foaf:mbox &lt;mailto:engineering-2112@a.example&gt; .</span></pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>reason</th> </tr> <tr class=pass> <td>&lt;issue8&gt;</td> <td>&lt;EmployeeShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=pass> <td>&lt;issue9&gt;</td> <td>&lt;EmployeeShape&gt;</td> <td>pass</td> <td class=noreason></td> </tr> <tr class=fail> <td>&lt;issue10&gt;</td> <td>&lt;EmployeeShape&gt;</td> <td>fail</td> <td class=fail><code>&lt;mailto:engineering-2112@a.example&gt;</code> is excluded.</td> </tr> </table> </div> <p> A value set can have a single value in it. This is used to indicate that a specific value is required, e.g. that an ex:state must be equal to &lt;http://schema.example/#Resolved&gt; or the rdf:type of some node must be foaf:Person. </p> </section> </section> <section id=shapes-and-TEs> <h3>Shapes and Triple Expressions</h3> <p> Triple expressions are used for defining patterns composed of triple constraints. Shapes associate <a>triple expressions</a> with flags indicating whether triples match if they do not correspond to triple constraints in the <a>triple expression</a>. A <dfn data-lt="triple expressions">triple expression</dfn> is composed of <span class=jobjref><a>TripleConstraint</a></span> and <span class=jobjref><a>tripleExprRef</a></span> objects composed with grouping and choice operators. </p> <section id=triple-expressions-abstract-syntax> <h4>JSON Syntax</h4> <div class=shexjTable> <table class=shexj> <tbody><tr class=obj> <th class=obj><dfn>Shape</dfn></th> <td>{</td> <td><span class=param>extends</span>:[<a class=objref>shapeExprRef</a>]? <span class=param>closed</span>:<a>BOOL</a>? <span class=param>extra</span>:[<a>IRIREF</a>+]? <span class=param>expression</span>:<a class=nobref>tripleExpr</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=nob> <th class=nob><dfn>tripleExpr</dfn></th> <td>=</td> <td><a class=objref>EachOf</a> | <a class=objref>OneOf</a> | <a class=objref>TripleConstraint</a> | <a class=objref>tripleExprRef</a> ;</td> </tr> <tr class=obj> <th class=obj><dfn>EachOf</dfn></th> <td>{</td> <td><span class=param>id</span>:<a class=nobref>tripleExprLabel</a>? <dfn class=param>expressions</dfn>:[<a class=nobref>tripleExpr</a>{2,}] <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=obj> <th class=obj><dfn>OneOf</dfn></th> <td>{</td> <td><span class=param>id</span>:<a class=nobref>tripleExprLabel</a>? <span class=param>expressions</span>:[<a class=nobref>tripleExpr</a>{2,}] <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=obj> <th class=obj><dfn>TripleConstraint</dfn></th> <td>{</td> <td><span class=param>id</span>:<a class=nobref>tripleExprLabel</a>? <span class=param>inverse</span>:<a>BOOL</a>? <a class=param>predicate</a>:<a>IRIREF</a> <span class=param>valueExpr</span>:<a class=nobref>shapeExpr</a>? <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=nob> <th class=nob><dfn>tripleExprRef</dfn></th> <td>=</td> <td><a class=objref>tripleExprLabel</a> ;</td> </tr> <tr class=nob> <th class=nob><dfn>tripleExprLabel</dfn></th> <td>=</td> <td><a>IRIREF</a> | <a>BNODE</a> ;</td> </tr> </table> </div> </section> <section id=triple-expressions-semantics> <h4>Semantics</h4> <p id=satisfies-Shape> The semantics of the <code>matchesShape</code> function are based on the <code>matches</code> function <a href=#matches>defined below</a>. A <a>shape</a> may have an <span class=param>expression</span>. For the purposes of evaluation, we define an <dfn>EmptyExpression</dfn> which has no <span class=jobjref><a class=obj>TripleConstraint</a>s</span>. </p> <p> <dfn>parentShapeLabels</dfn> is a function from a <a>shape</a> <span class=var>label</span> to the set of <a>shapeExprLabels</a> parents of the labels in shape.extends as well as their parents in the <a>extension hierarchy graph</a>. </p> <p> For a <a>node</a> <span class=math>n</span>, <a>shape</a> <span class=math>S</span>, <a>graph</a> <span class=math>G</span>, a ShExSchema <span class=math>Sch</span>, a <a>typing</a> <span class=math>m</span>, and a subset <span class=math>R</span> of neigh(n), <code class=function><dfn>matchesShape</dfn>(<span class=math>n</span>, <span class=math>S</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>, <span class=math>R</span>)</code> if and only if: </p> <ul> <li><dfn>parents</dfn> is the set parentShapeLabels(S). If s.extends does not exist, then parentShapeLabels(s) is the empty set. </li> <li><dfn class=math>nCard</dfn> is the length of parents.</li> <li> R can be partitioned into two sets <span class=math>matched</span> and <span class=math>remainder</span></li> <li><span class=math>matched</span> is partitioned into <code class=math>nCard+1</code> parts <code class=math>R<sub>0</sub></code>, <code class=math>R<sub>1</sub></code> ... <code class=math>R<sub>nCard</sub></code> such that <ul> <li><code class=function>matches(<span class=math>R<sub>0</sub></span>, S.tripleExpr, <span class=math>m</span>)</code> </li> <li>satisfies(n, constraint(L), G, Sch, m, matched)</li> <li> for every i in 1..nCard, <code class=function>matches(<span class=math>R<sub>i</sub></span>, mainShape(parents<sub>i</sub>), <span class=math>m</span>)</code> </li> <li> for every i in 1..nCard, satisfies(n, constraint(parent<sub>i</sub>), G, Sch, m, R<sub>i</sub> Ôê¬ Q) where Q is the union of all the R<sub>j</sub> s.t. parent<sub>j</sub> is a parent of parent<sub>i</sub>.</li> </ul> </li> <li> Let <span class=label>outs</span> be the <span class=math><a>arcsOut</a></span> in <span class=math>remainder</span>: <code class=function><span class=math>outs</span> = <span class=math>remainder</span> Ôê® <a>arcsOut</a>(<span class=math>G</span>, <span class=math>n</span>)</code>. </li> <li> Let <span class=label>matchables</span> be the triples in <span class=math>outs</span> whose <a>predicate</a> appears in a <span class=jobjref><a class=obj>TripleConstraint</a></span> in one of the mainShape(parents<sub>i</sub>) or in S.expression. </li> <li> There is no triple in <span class=label>matchables</span> which matches a <span class=jobjref><a class=obj>TripleConstraint</a></span> in one of the mainShape(parents<sub>i</sub>) nor one of the TripleConstraint in S.expression.<br> Let <span class=label>unmatchables</span> be the triples in <span class=math>outs</span> which are not in <span class=math>matchables</span>. <code class=function><span class=math>matchables</span> Ôê¬ <span class=math>unmatchables</span> = <span class=math>outs</span></code>. </li> <li> There is no triple in <span class=math>matchables</span> whose <a>predicate</a> does not appear in <span class=param>extra</span>. </li> <li><span class=param>closed</span> is false or <span class=math>unmatchables</span> is empty. </li> <p class=note> The complexity of partitioning is described briefly in the <a href=file:///C:/src/word/HTML2Doc/primer/#partition-complexity>ShEx2 Primer</a>. </p> </ul> <p> <span class=label id=matches>matches</span>: asserts that a <a>triple expression</a> is matched by a set of triples that come from the neighbourhood of a node in an <a>RDF graph</a>. The expression <code>matches(<span class=math>T</span>, <span class=math>expr</span>, <span class=math>m</span>)</code> indicates that a set of triples <span class=math>T</span> can satisfy these rules: </p> <ul> <li> <p id=matches-semActs><span class=math>expr</span> has <span class=param>semActs</span> and <code class=function>matches(<span class=math>T</span>, <span class=math>expr</span>, <span class=math>m</span>)</code> by the remaining rules in this list and the evaluation of <span class=param>semActs</span> succeeds according to the section below on <a href=#semantic-actions>Semantic Actions</a>. </p> <div class=example> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "OneOf", "shapeExprs": [<span class=comment>te<sub>1</sub></span>, <span class=comment>te<sub>2</sub>, ÔÇª</span>], "min": 2, "max": 3, "semActs": [<span class=comment>SemAct<sub>1</sub></span>, <span class=comment>SemAct<sub>2</sub>, ÔÇª</span>] }</pre> <pre class="shexc nohighlight">(<span class=comment>te<sub>1</sub></span> | <span class=comment>te<sub>2</sub>, ÔÇª</span>) {2,3} &nbsp; %&lt;SemAct<sub>1</sub>&gt;% %&lt;SemAct2&gt;% ÔÇª </pre> </div>,<br> m) </div> </div> <div style=clear:both>evaluates as:</div> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "OneOf", "shapeExprs": [<span class=comment>te<sub>1</sub></span>, <span class=comment>te<sub>2</sub>, ÔÇª</span>], "min": 2, "max": 3 } &nbsp;</pre> <pre class="shexc nohighlight">(<span class=comment>te<sub>1</sub></span> | <span class=comment>te<sub>2</sub>, ÔÇª</span>) {2,3} &nbsp;</pre> </div>,<br> m) </div> </div> <div style=clear:both></div> <span class="math function">and <a href=#semActsSatisfied>semActsSatisfied</a>(<span class="inline json">[<span class=comment>SemAct<sub>1</sub></span>, <span class=comment>SemAct<sub>2</sub>, ÔÇª</span>]</span>)</span> </div> </li> <li> <p id=matches-cardinality><span class=math>expr</span> has a <dfn>cardinality</dfn> of <span class=param>min</span> and/or <span class=param>max</span> not equal to 1, where a <span class=param>max</span> of -1 is treated as unbounded, and <span class=math>T</span> can be partitioned into <span class=math>k</span> subsets <span class="math function">T<sub>1</sub></span>, <span class=math>T<sub>2</sub></span>,ÔÇª<span class=math>T<sub>k</sub></span> such that <span class=math>min Ôëñ k Ôëñ max</span> and for each <span class=math>T<sub>n</sub></span>, <code class=function>matches(<span class=math>T<sub>n</sub></span>, <span class=math>expr</span>, <span class=math>m</span>)</code> by the remaining rules in this list.</p> <div class=example> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "OneOf", "shapeExprs": [<span class=comment>te<sub>1</sub></span>, <span class=comment>te<sub>2</sub>, ÔÇª</span>], "min": 2, "max": 3 } &nbsp;</pre> <pre class="shexc nohighlight">(<span class=comment>te<sub>1</sub></span> | <span class=comment>te<sub>2</sub>, ÔÇª</span>) {2,3} &nbsp;</pre> </div>,<br> m) </div> </div> <div style=clear:both>evaluates as:</div> <div class=math> <div style=float:left>Let e =&nbsp;</div> <div style=float:left> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "OneOf", "shapeExprs": [<span class=comment>te<sub>1</sub></span>, <span class=comment>te<sub>2</sub>, ÔÇª</span>] } &nbsp;</pre> <pre class="shexc nohighlight">(<span class=comment>te<sub>1</sub></span> | <span class=comment>te<sub>2</sub>, ÔÇª</span>) &nbsp;</pre> </div> </div> <div style=clear:both></div> (<span class="math function">matches(T<sub>1</sub>, e, m)</span> and <span class="math function">matches(T<sub>2</sub>, e, m)</span><br> &nbsp;and T = T<sub>1</sub> Ôê¬ T<sub>2</sub>)<br> or<br> (<span class="math function">matches(T<sub>1</sub>, e, m)</span> and <span class="math function">matches(T<sub>2</sub>, e, m)</span> and <span class="math function">matches(T<sub>3</sub>, e, m)</span><br> &nbsp;and T = T<sub>1</sub> Ôê¬ T<sub>2</sub> Ôê¬ T<sub>3</sub>) </div> </div> </li> <li> <p id=matches-OneOf><span class=math>expr</span> is a <span class=jobjref><a class=obj>OneOf</a></span> and there is some <a>shape expression</a> <span class=math>se2</span> in <span class=param>shapeExprs</span> such that <code class=function>matches(<span class=math>T</span>, <span class=math>se2</span>, <span class=math>m</span>)</code>. </p> <div class=example> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "OneOf", "shapeExprs": [ { "type": "EachOf", "shapeExprs": [<span class=comment>te<sub>3</sub></span>, <span class=comment>te<sub>4</sub>, ÔÇª</span>] }, { "type": "TripleExpression", "min": 1, "max": -1, "predicate": "http://xmlns.com/foaf/0.1/name" } ] }</pre> <pre class="shexc nohighlight"> &nbsp; (<span class=comment>te<sub>3</sub></span> ; <span class=comment>te<sub>4</sub> ; ÔÇª</span>) | &lt;http://xmlns.com/foaf/0.1/name&gt; . + &nbsp;</pre> </div>,<br> m) </div> </div> <div style=clear:both>evaluates as:</div> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "EachOf", "shapeExprs": [<span class=comment>te<sub>3</sub></span>, <span class=comment>te<sub>4</sub>, ÔÇª</span>] } &nbsp;</pre> <pre class="shexc nohighlight"><span class=comment>te<sub>3</sub></span> ; <span class=comment>te<sub>4</sub> ; ÔÇª</span> &nbsp;</pre> </div>,<br> m) </div> </div> <div style=clear:both></div> <div class=math> <div style=float:left>or matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "TripleExpression", "min": 1, "max": -1, "predicate": "http://xmlns.com/foaf/0.1/name" }</pre> <pre class="shexc nohighlight">&lt;http://xmlns.com/foaf/0.1/name&gt; . + &nbsp; &nbsp;</pre> </div>,<br> m) </div> </div> </div> </li> <li> <p id=matches-EachOf><span class=math>expr</span> is an <span class=jobjref><a class=obj>EachOf</a></span> and there is some partition of <span class=math>T</span> into <span class=math>T<sub>1</sub></span>, <span class=math>T<sub>2</sub></span>,ÔÇª such that for every expression <span class=function><span class=math>expr<sub>1</sub></span>, <span class=math>expr<sub>2</sub></span>,ÔÇª</span> in <span class=param>shapeExprs</span>, <code class=function>matches(<span class=math>T<sub>n</sub></span>, <span class=math>expr<sub>n</sub></span>, <span class=math>m</span>)</code>. </p> <div class=example> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "EachOf", "shapeExprs": [ { "type": "TripleExpression", "predicate": "http://xmlns.com/foaf/0.1/givenName" }, { "type": "TripleExpression", "predicate": "http://xmlns.com/foaf/0.1/familyName" } ] }</pre> <pre class="shexc nohighlight"> &nbsp; &lt;http://xmlns.com/foaf/0.1/givenName&gt; . ; &lt;http://xmlns.com/foaf/0.1/familyName&gt; . &nbsp;</pre> </div>,<br> m) </div> </div> <div style=clear:both>evaluates as:</div> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T<sub>1</sub>,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "TripleExpression", "predicate": "http://xmlns.com/foaf/0.1/givenName" }</pre> <pre class="shexc nohighlight">&lt;http://xmlns.com/foaf/0.1/givenName&gt; . &nbsp; &nbsp;</pre> </div>,<br> m) </div> </div> <div style=clear:both></div> <div class=math> <div style=float:left>and matches(</div> <div style=float:left>T<sub>2</sub>,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "TripleExpression", "predicate": "http://xmlns.com/foaf/0.1/familyName" }</pre> <pre class="shexc nohighlight">&lt;http://xmlns.com/foaf/0.1/familyName&gt; . &nbsp; &nbsp;</pre> </div>,<br> m) </div> </div> <div style=clear:both></div> <span class=math>and T = T<sub>1</sub> Ôê¬ T<sub>2</sub></span> </div> </li> <li> <p id=matches-TripleConstraint><span class=math>expr</span> is a <span class=jobjref><a class=obj>TripleConstraint</a></span> and:</p> <ul> <li> <span class=math>T</span> is a set of one triple.<br> Let <span class=label>t</span> be the sole triple in <span class=math>T</span>. </li> <li> <span class=math>t</span>'s <a class=param>predicate</a> equals <span class=math>expr</span>'s <a>predicate</a>.<br> Let <span class=label>value</span> be <span class=math>t</span>'s subject if <span class=param>inverse</span> is true, else <span class=math>t</span>'s object. </li> <li> if <span class=param>inverse</span> is true, <span class=math>t</span> is in <span class=math><a>arcsIn</a></span>, else <span class=math>t</span> is in <span class=math><a>arcsOut</a></span>. </li> <li>either <ul> <li> <p><span class=math>expr</span> has no <span class=param>valueExpr</span> </p> <div class=example> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "TripleExpression", "predicate": "http://xmlns.com/foaf/0.1/givenName" }</pre> <pre class="shexc nohighlight">&lt;http://xmlns.com/foaf/0.1/givenName&gt; . ; &nbsp; &nbsp;</pre> </div>,<br> m) </div> </div> <div style=clear:both>holds if <ul class=tight> <li><span class=math>T</span> has exactly one triple <span class=math>t</span>.</li> <li><span class=math>t</span> has the <a>predicate</a> "<code>http://xmlns.com/foaf/0.1/givenName</code>"</li> </ul> </div> </div> </li> <li>or expr.valueExpr is a shapeExprRef, then shapeExprRef.label is in m(value) </li> <li> <p>or expr.valueExpr is not a shapeExprRef, then <code class=function>satisfies(<span class=math>value</span>, <span class=param>valueExpr</span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>, neigh(value))</code>. </p> <div class=example> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline repchoice incomplete"> <pre class="json nohighlight">{ "type": "TripleConstraint", "inverse": true, "predicate": "http://purl.org/dc/elements/1.1/author", "valueExpr": "http://schema.example/#IssueShape" }</pre> <pre class="shexc nohighlight"> &nbsp; ^&lt;http://purl.org/dc/elements/1.1/author&gt; @&lt;http://schema.example/#IssueShape&gt;</pre> </div>,<br> m) </div> </div> <div style=clear:both>holds if <ul class=tight> <li><span class=math>T</span> has exactly one triple <span class=math>t</span>.</li> <li><span class=math>t</span> has the <a>predicate</a> "<code>http://purl.org/dc/elements/1.1/author</code>"</li> <li><span class=math>t</span> has a subject <span class=math>n2</span></li> <li>The schema's <a class=param>shapes</a> maps "<code>http://schema.example/#IssueShape</code>" to <span class=math>se2</span></li> <li><span class="math function">satisfies(n2, se2, G, Sch, m)</span> </li> </ul> </div> </div> </li> </ul> </li> </ul> </li> <li> <p id=matches-tripleExprRef> <span class=math>expr</span> is a <span class=jobjref><a class=obj>tripleExprRef</a></span> and <code class=function>satisfies(<span class=math>value</span>, tripleExprWithId(<span class=param>tripleExprRef</span>), <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>Sch</span>, <span class=math>m</span>)</code>.<br> The <code class=function><a>tripleExprWithId</a></code> function is defined in <a href=#tripleExprRef-requirement>Triple Expression Reference Requirement</a> below. </p> <div class=example> For the schema<br> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#EmployeeShape", "type": "Shape", "expression": { "type": "EachOf", "expressions": [ "http://schema.example/#nameExpr", { "type": "TripleConstraint", "predicate": "http://schema.example/#empID", "valueExpr": { "type": "NodeConstraint", "datatype": "http://www.w3.org/2001/XMLSchema#integer" } } ] } }, { "id": "http://schema.example/#PersonShape", "type": "Shape", "expression": { "id": "http://schema.example/#nameExpr", "type": "TripleConstraint", "predicate": "http://xmlns.com/foaf/0.1/name" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; &lt;http://schema.example/#EmployeeShape&gt; { &amp;<span class=lookit>&lt;http://schema.example/#nameExpr&gt;</span> ; &lt;http://schema.example/#empID&gt; &lt;http://www.w3.org/2001/XMLSchema#integer&gt; } &lt;http://schema.example/#PersonShape&gt; { <span class=lookit>$&lt;http://schema.example/#nameExpr&gt;</span> &lt;http://xmlns.com/foaf/0.1/name&gt; . ; }</pre> </div> <div style=clear:both></div> <div class=math> <div style=float:left>matches(</div> <div style=float:left>T,<br> <div class="inline json"> <pre>"http://schema.example/#PersonShape"</pre> </div>,<br> m) </div> </div> <div style=clear:both>holds if <ul class=tight> <li>The schema has a shape <span class=math>se2</span> with the <span class=param>id</span> "<code>http://schema.example/#PersonShape</code>" </li> <li><span class="math function">satisfies(n, se2, G, Sch, m)</span></li> </ul> </div> </div> </li> </ul> </section> </section> <section id=import> <h3>ShEx Import</h3> <p> The presence of <span class=param>imports</span> requires that: </p> <ul> <li>each IRI in <span class=param>imports</span> be resolved and</li> <li>the returned representation of that IRI be interpreted as a ShEx <span class=math>S</span> and </li> <li>each <a class=jobjref>shapeExpr</a> in <span class=math>S</span>.<span class=param>shapes</span> be in scope for resolving shape expression references and</li> <li>each <a class=jobjref>tripleExpr</a> with a <a class=jobjref>tripleExprLabel</a> be in scope for resolving triple expression references.</li> </ul> <p> If any imported schema imports other schemas, shape and triple expression labels from those schemas are also in scope. </p> <div class=example> <div class="example-title marker"><span>Import example 1 - Shape and Triple Expressions</span></div> <div class="repchoice incomplete"> <pre class="json nohighlight"><span class=label>schema1:</span> { "type": "Schema", "imports": ["http://schema.example/schema2"], "shapes": [ { "id": "http://schema.example/#EmployeeShape", "type": "Shape", "expression": { "type": "EachOf", "expressions": [ "http://schema.example/#nameExpr", { "type": "TripleConstraint", "predicate": "http://schema.example/#empID", "valueExpr": { "type": "NodeConstraint", "datatype": "http://www.w3.org/2001/XMLSchema#integer" } } ] } } ] } <span class=label>schema2:</span> { "type": "Schema", "shapes": [ { "id": "http://schema.example/#PersonShape", "type": "Shape", "expression": { "id": "http://schema.example/#nameExpr", "type": "TripleConstraint", "predicate": "http://xmlns.com/foaf/0.1/name" } } ] } </pre> <pre class="shexc nohighlight"><span class=label>schema1:</span> &nbsp; &lt;http://schema.example/#EmployeeShape&gt; { &amp;&lt;http://schema.example/#nameExpr&gt; ; &lt;http://schema.example/#empID&gt; &lt;http://www.w3.org/2001/XMLSchema#integer&gt; } <span class=label>schema2:</span> <span class=lookit>&lt;http://schema.example/#PersonShape&gt;</span> { <span class=lookit>$&lt;http://schema.example/#nameExpr&gt;</span> &lt;http://xmlns.com/foaf/0.1/name&gt; ; } &nbsp;</pre> </div> <p> Both the shape expression <code>&lt;PersonShape&gt;</code> and the triple expression <code>&lt;nameExpr&gt;</code> are in scope.<br> <span class=label>schema2</span>'s <code>&lt;nameExpr&gt;</code> is referenced in <span class=label>schema1</span>'s <code>&lt;EmployeeShape&gt;</code> </p> </div> <p> Redundant imports are treated as a single import. This includes circular imports: </p> <div class=example> <div class="example-title marker"><span>Import example 2 - Circular Import</span></div> <div class="repchoice incomplete"> <pre class="json nohighlight"><span class=label>schema1:</span> { "type": "Schema", "imports": ["http://schema.example/schema2", "http://schema.example/schema3"], "shapes": [ { "id": "http://schema.example/schema1#S1", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p1", "valueExpr": "http://schema.example/schema1#S2" } } ] } <span class=label>schema2:</span> { "type": "Schema", "imports": ["http://schema.example/schema3"], "shapes": [ { "id": "http://schema.example/schema1#S2", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p2", "valueExpr": "http://schema.example/schema1#S3" } } ] } <span class=label>schema3:</span> { "type": "Schema", "imports": ["http://schema.example/schema1"], "shapes": [ { "id": "http://schema.example/schema1#S3", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p3", "valueExpr": "http://schema.example/schema1#S1", "min": 0, } } ] } </pre> <pre class="shexc nohighlight"><span class=label>schema1:</span> &nbsp; IMPORT &lt;http://schema.example/schema2&gt; IMPORT &lt;http://schema.example/schema3&gt; &lt;http://schema.example/schema1#S1&gt; { &lt;http://schema.example/#p1&gt; @&lt;http://schema.example/schema1#S2&gt; <span class=label>schema2:</span> IMPORT &lt;http://schema.example/schema3&gt; &lt;http://schema.example/schema1#S2&gt; { &lt;http://schema.example/#p2&gt; @&lt;http://schema.example/schema1#S3&gt; <span class=label>schema3:</span> IMPORT &lt;http://schema.example/schema1&gt; &lt;http://schema.example/schema1#S3&gt; { &lt;http://schema.example/#p3&gt; @&lt;http://schema.example/schema1#S1&gt;? &nbsp;</pre> </div> </div> <p> When some schema <span class=math>A</span> imports schema <span class=math>B</span>, <span class=math>B</span>'s <a class=jobjref>start</a> member is ignored. </p> <div class=example> <div class="example-title marker"><span>Import example 3 - Ignored Start In Import</span></div> <div class="repchoice incomplete"> <pre class="json nohighlight"><span class=label>schema1:</span> { "type": "Schema", "imports": ["http://schema.example/schema2"], "shapes": [ { "id": "http://schema.example/schema1#S1", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p1", "valueExpr": "http://schema.example/schema1#S2" } } ] } <span class=label>schema2:</span> { "type": "Schema", <span class=lookit>"start": "http://schema.example/schema1#S2"</span>, "shapes": [ { "id": "http://schema.example/schema1#S2", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p2" } } ] } </pre> <pre class="shexc nohighlight"><span class=label>schema1:</span> &nbsp; IMPORT &lt;http://schema.example/schema2&gt; &lt;http://schema.example/schema1#S1&gt; { &lt;http://schema.example/#p1&gt; @&lt;http://schema.example/schema1#S2&gt; } <span class=label>schema2:</span> <span class=lookit>start=@&lt;http://schema.example/schema1#S2&gt;</span> &lt;http://schema.example/schema1#S2&gt; { &lt;http://schema.example/#p2&gt; . } </pre> </div> <p> <code>schema1</code> has no <code>start</code> even though it imports a schema with a <code>start</code>. </p> </div> <p> It is an error if <code>A</code> and <code>B</code> share any labels for shape expressions or triple expressions or if schema <code>B</code> has a <a class=jobjref>startActs</a> member. </p> <div class=example> <div class="example-title marker"><span>Import example 4 - Erroneous Import</span></div> <div class="repchoice error incomplete"> <pre class="json nohighlight"><span class=label>schema1:</span> { "type": "Schema", "imports": ["http://schema.example/schema2"], "shapes": [ { "id": "http://schema.example/schema1#S1", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p1", "valueExpr": "http://schema.example/schema1#S2" } } ] } <span class=label>schema2:</span> { "type": "Schema", <span class=lookit>"startActs": [ { "type": "semAct", "name": "http://schema.example/schema1#A1" } ]</span>, "shapes": [ { <span class=lookit>"id": "http://schema.example/schema1#S1"</span>, "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p1", "valueExpr": "http://schema.example/schema1#S2" } }, { "id": "http://schema.example/schema1#S2", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p2", "valueExpr": <span class=lookit>"http://schema.example/schema1#S3"</span> } } ] } </pre> <pre class="shexc nohighlight"><span class=label>schema1:</span> &nbsp; IMPORT &lt;http://schema.example/schema2&gt; &lt;http://schema.example/schema1#S1&gt; { &lt;http://schema.example/#p1&gt; @&lt;http://schema.example/schema1#S2&gt; } <span class=label>schema2:</span> <span class=lookit>%@&lt;http://schema.example/schema1#A1&gt;%</span> <span class=lookit>&lt;http://schema.example/schema1#S1&gt;</span> { &lt;http://schema.example/#p1&gt; @&lt;http://schema.example/schema1#S2&gt; } &lt;http://schema.example/schema1#S2&gt; { &lt;http://schema.example/#p2&gt; <span class=lookit>@&lt;http://schema.example/schema1#S3&gt;</span> } </pre> </div> <p> This import fails because: </p> <ul> <li><code>&lt;http://schema.example/schema1#S1&gt;</code> has conflicting definitions and</li> <li>an included schema has a <code>start</code> directive and</li> <li>the reference to <code>&lt;http://schema.example/schema1#S3&gt;</code> is not resolvable after imports.</li> </ul> </div> </section> <section id=schema-requirements> <h3>Schema Requirements</h3> <p> The semantics defined above assume three structural requirements beyond those imposed by the grammar of the abstract syntax. These ensure referential integrity and eliminate logical paradoxes such as those that arrise through the use of negation. These are not constraints expressed by the schema but instead those imposed on the schema. </p> <section id=validation-requirement> <h4>Schema Validation Requirement</h4> <p>A <a>graph</a> <span class=math>G</span> is said to <dfn data-lt=conforms|conformance>conform</dfn> with a <a>schema</a> <span class=math>S</span> with a <a>ShapeMap</a> <span class=math>m</span> when: </p> <ol class=algorithm> <li>Every, <a>SemAct</a> in the <a>startActs</a> of <span class=math>S</span> has a successful evaluation of <a href=#semActsSatisfied>semActsSatisfied</a>.</li> <li>Every <a>node</a> <span class=math>n</span> in <span class=math>m</span> <a>conforms</a> to its associated <a>shapeExprRefs</a> <span class=math>se<sub>n</sub></span> where for each <a>shapeExprRef</a> <span class=math>se<sub>i</sub></span> in <span class=math>se<sub>n</sub></span>: <ol class=algorithm> <li><span class=math>se<sub>i</sub></span> references a <a>ShapeExpr</a> in <a>shapes</a>, and</li> <li><code class=function><a>satisfies</a>(<span class=math>n</span>, <span class=math>se<sub>i</sub></span>, <span class=math>G</span>, <span class=math>Sch</span>, <span class=math>m</span>)</code> for each <a>shape</a> <span class=math>se<sub>i</sub></span> in <span class=math>se<sub>n</sub></span>.</li> </ol> </li> </ol> </section> <section id=shapeExprRef-requirement> <h4>Shape Expression Reference Requirement</h4> <p>A <span class=jobjref><a>shapeExprRef</a></span> MUST appear in the schema's <a class=param>shapes</a> map (or an <a href=#import>imported schema's</a> map) and the corresponding <a>shape expression</a> MUST be a <span class=jobjref><a>Shape</a></span> with a <span class=param>shapeExpr</span>. The function <code>shapeExprWithId(<span class=math>shapeExprRef</span>)</code> returns the shape expression with an <span class=param>id</span> of <span class=math>shapeExprRef</span>. </p> <p> Additionally, a <span class=jobjref><a>shapeExprLabel</a></span> cannot refer to itself through a shape reference either directly or recursively. The <dfn>shapeExprRef closure</dfn> of a <a>shape expression</a> <span class=math>se</span> is the set of shape expression labels used as references in <span class=math>se</span>. The <a>shapeExprLabel</a> <span class=math>sl</span> belongs to <code class=function>shapeExprRefClosure(<span class=math>se</span>)</code> if and only if: </p> <ul> <li> <span class=math>sl</span> appears as an atomic <span class=jobjref><a>shapeExprRef</a></span> in <span class=math>se</span>, or </li> <li> <span class=math>sl</span> belongs to <code class=function>shapeExprRefClosure(shapeExprWithId(<span class=math>sl2</span>))</code> for some <a>shapeExprLabel</a> <span class=math>sl2</span> that belongs to <code class=function>shapeExprRefClosure(<span class=math>se</span>)</code>. </li> </ul> <p> A shapes schema MUST NOT define a shape label <span class=math>sl</span> that belongs to the shapeExprRef closure of its definition <code class=function>shapeExprWithId(<span class=math>sl</span>)</code>. </p> <div class=example> <p> Following are two valid shapeExprRefs: </p> <div class=repchoice> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id" : "http://schema.example/#PersonShape", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://xmlns.com/foaf/0.1/name" } }, { "id" : "http://schema.example/#EmployeeShape", "type" : "ShapeAnd", "shapeExprs" : [ "http://schema.example/#PersonShape", { "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#employeeNumber" } } ] } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:PersonShape { foaf:name . } ex:EmployeeShape @ex:PersonShape AND { ex:employeeNumber . } &nbsp; </pre> </div> <div class=repchoice> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id" : "http://schema.example/#PersonShape", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://xmlns.com/foaf/0.1/name" } }, { "id" : "http://schema.example/#EmployeeShape", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#dependent", "valueExpr" : "http://schema.example/#PersonShape", "min" : 0, "max" : -1 } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:PersonShape { foaf:name . } ex:EmployeeShape { ex:dependent @ex:PersonShape \* } &nbsp; </pre> </div> <p> This shapeExprRef is invalid because there is no corresponding <a>shape expression</a>: </p> <div class="repchoice error"> <pre class="json nohighlight">{ "type":"Schema", "shapes": [ { "id": "http://schema.example/#S1", "type":"Shape", "expression": "http://schema.example/#MissingShapeExpr" } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:S1 { &amp;ex:MissingShapeExpr }</pre> </div> <p>This shapeExprRef is invalid because the referenced object is a <a>triple expression</a> instead of a <a>shape expression</a>:</p> <div class="repchoice error"> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id" : "http://schema.example/#CustomerShape", "type" : "Shape", "expression" : { "id" : "http://schema.example/#discountExpr", "type" : "TripleConstraint", "predicate" : "http://schema.example/#discount" } }, { "id" : "http://schema.example/#EmployeeShape", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#contactFor", "valueExpr" : "http://schema.example/#discountExpr" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:CustomerShape { $ex:discountExpr ex:discount . } ex:EmployeeShape { ex:contactFor @ex:discountExpr } &nbsp; </pre> </div> <p>These shapeExprRefs are invalid because they recursively refer to each other.</p> <div class="repchoice error"> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id" : "http://schema.example/#PersonShape", "type" : "ShapeAnd", "shapeExprs" : [ "http://schema.example/#EmployeeShape", { "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://xmlns.com/foaf/0.1/name" } } ] }, { "id" : "http://schema.example/#EmployeeShape", "type" : "ShapeAnd", "shapeExprs" : [ "http://schema.example/#PersonShape", { "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#employeeNumber" } } ] } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:PersonShape @ex:EmployeeShape AND { foaf:name . } ex:EmployeeShape @ex:PersonShape AND { ex:employeeNumber . } &nbsp; </pre> </div> </div> </section> <section id=tripleExprRef-requirement> <h4>Triple Expression Reference Requirement</h4> <p>An <span class=jobjref><a>tripleExprRef</a></span> MUST identify a <a>triple expression</a> in the schema. The function <code><dfn>tripleExprWithId</dfn>(<span class=math>tripleExprRef</span>)</code> returns the <a>triple expression</a> with the id <span class=math>tripleExprRef</span>. </p> <p> Additionally, a <span class=jobjref><a>tripleExprLabel</a></span> cannot refer to itself through a triple expression reference either directly or recursively. The <dfn>tripleExprRef closure</dfn> of a <a>triple expression</a> <span class=math>te</span> is the set of triple expression labels used as references in <span class=math>te</span>. The <a>tripleExprLabel</a> <span class=math>tl</span> belongs to <code class=function>tripleExprRefClosure(<span class=math>te</span>)</code> if and only if: </p> <ul> <li> <span class=math>tl</span> appears as an atomic <span class=jobjref><a>tripleExprRef</a></span> in <span class=math>te</span>, or </li> <li> <span class=math>tl</span> belongs to <code class=function>tripleExprRefClosure(tripleExprWithId(<span class=math>tl2</span>))</code> for some <a>tripleExprLabel</a> <span class=math>tl2</span> that belongs to <code class=function>tripleExprRefClosure(<span class=math>te</span>)</code>. </li> </ul> <p> A shapes schema MUST NOT define a triple expression label <span class=math>tl</span> that belongs to the tripleExprRef closure of its definition <code class=function>tripleExprWithId(<span class=math>tl</span>)</code>. </p> <div class=example> <p> Following is a valid <a>triple expression</a> reference: </p> <div class=repchoice> <pre class="json nohighlight">{ "type":"Schema", "shapes": [ { "id": "http://schema.example/#PersonShape", "type":"Shape", "expression": { "id": "http://schema.example/#nameExpr", "type": "TripleConstraint", "predicate": "http://xmlns.com/foaf/0.1/name" } }, { "id": "http://schema.example/#EmployeeShape", "type":"Shape", "expression": { "type":"EachOf", "expressions": [ "http://schema.example/#nameExpr", { "type": "TripleConstraint", "predicate": "http://schema.example/#employeeNumber" } ] } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:PersonShape { $ex:nameExpr foaf:name . } ex:EmployeeShape { &amp;ex:nameExpr ; ex:employeeNumber . } &nbsp; </pre> </div> <p> This <a>triple expression</a> reference is invalid because there is no corresponding triple expression: </p> <div class="repchoice error"> <pre class="json nohighlight">{ "type":"Schema", "shapes": [ { "id": "http://schema.example/#S1", "type":"Shape", "expression": "http://schema.example/#missingTripleExpr" } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:S1 { &amp;ex:missingTripleExpr } &nbsp; </pre> </div> <p>This triple expression reference is invalid because the referenced object is a <a>shape expression</a> instead of a triple expression:</p> <div class="repchoice error incomplete"> <pre class="json nohighlight">{ "type":"Schema", "shapes": [ { "id": "http://schema.example/#CustomerShape", "type":"ShapeAnd", "shapeExprs": [ <span class=comment>ÔÇª</span> ] }, { "id": "http://schema.example/#PreferredCustomerShape", "type":"Shape", "expression": { "type":"EachOf", "expressions": [ "http://schema.example/#CustomerShape", { "type": "TripleConstraint", "predicate": "http://schema.example/#discount" } ] } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:CustomerShape { <span class=comment>ÔÇª</span>; <span class=comment>ÔÇª</span> } ex:PreferredCustomerShape { &amp;ex:CustomerShape ; ex:discount . } &nbsp; </pre> </div> </div> </section> <section id=shapeExprRef-non-abstract-shape-requirement> <h4>shapeExprRef non-abstract shape requirement</h4> <p> Every <a>shapeExprRef</a> |referer| MUST identify at least one non-abstract shape. </p> <div class=example> <p> Following is a valid example with a shape with a <a>shapeExprRef</a> that references an abstract shape with two non-abstract descendants: </p> <div class=repchoice> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#approvedBy", "valueExpr": "http://schema.example/#EngineerShape" } }, { "id" : "http://schema.example/#EntityShape", "type" : "Shape", "abstract": true, "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#entityId" } }, { "id" : "http://schema.example/#PersonShape", "type" : "Shape", "extends" : [ "http://schema.example/#EntityShape" ], "expression" : { "type" : "TripleConstraint", "predicate" : "http://xmlns.com/foaf/0.1/name" } }, { "id" : "http://schema.example/#EmployeeShape", "type" : "Shape", "extends" : [ "http://schema.example/#PersonShape" ], "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#employeeNumber" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:approvedBy @ex:EntityShape } ABSTRACT ex:EntityShape { ex:entityId . } ex:PersonShape EXTENDS @ex:EntityShape { foaf:name . } ex:EmployeeShape EXTENDS @ex:EntityShape { ex:employeeNumber . } &nbsp; </pre> </div> <p> This <a>shapeExprRef</a> is invalid because it references only abstract descendants: </p> <div class="repchoice error"> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#approvedBy", "valueExpr": "http://schema.example/#EngineerShape" } }, { "id" : "http://schema.example/#EntityShape", "type" : "Shape", "abstract": true, "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#entityId" } }, { "id" : "http://schema.example/#PersonShape", "type" : "Shape", "abstract": true, "extends" : [ "http://schema.example/#EntityShape" ], "expression" : { "type" : "TripleConstraint", "predicate" : "http://xmlns.com/foaf/0.1/name" } }, { "id" : "http://schema.example/#EmployeeShape", "type" : "Shape", "abstract": true, "extends" : [ "http://schema.example/#PersonShape" ], "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#employeeNumber" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:approvedBy @ex:EntityShape } ABSTRACT ex:EntityShape { ex:entityId . } ABSTRACT ex:PersonShape EXTENDS @ex:EntityShape { foaf:name . } ABSTRACT ex:EmployeeShape EXTENDS @ex:EntityShape { ex:employeeNumber . } &nbsp; </pre> </div> </div> </section> <section id=negation-requirement> <h4>Negation Requirement</h4> <p> A schema MUST NOT contain any <a>shapeExprLabel</a> that has a <a>negated reference</a> to itself, either directly or transitively. This is formalized by the requirement that the <a>hierarchy and dependency graph</a> of a schema MUST NOT have a cycle that traverses some <a>negated reference</a>. </p> <p class=leadup> The set of <dfn>atomic shapes</dfn> of a <a>shapeExpr</a> <span class=math>se</span> contains a <a>Shape</a> <span class=math>s</span> if <span class=math>s</span> or its <span class=path>id</span> appears either directly or by <a>shapeExprRef</a> in <span class=math>se</span>. That is, <span class=math>s</span> belongs to <code class=function>atomicShapes(<span class=math>se</span>)</code> if and only if </p> <ul class="followup leadup"> <li> <span class=math>s</span> appears as an atomic shape in <span class=math>se</span>, or </li> <li> <span class=math>sid</span> is the <span class=param>id</span> of <span class=math>s</span> and <span class=math>sid</span> appears as an atomic shapeExprRef in <span class=math>se</span>, or </li> <li> <span class=math>s</span> belongs to <code>atomicShapes(se2)</code> for some shape expression <span class=math>se2</span> such that the <span class=param>id</span> of <span class=math>se2</span> belongs to the shapeExprRefClosure of <span class=math>se</span>. </li> </ul> <p class=leadup>The set of <dfn>atomicTripleConstraints</dfn> of a <a>tripleExpr</a> <span class=math>te</span> includes every <a>TripleConstraint</a> <span class=math>tc</span> that appears directly or by <a>tripleExprRef</a> in <span class=math>te</span>. That is, <span class=math>tc</span> belongs to <code class=function><a>atomicTripleConstraints</a>(<span class=math>te</span>)</code> if and only if: </p> <ul> <li> <span class=math>tc</span> is an atomic <a>TripleConstraint</a> in <span class=math>te</span>, or </li> <li> <span class=math>te</span> is an atomic <a>TripleConstraint</a> in <code class=function>tripleExprWithId(<span class=math>tl</span>)</code> for some <a>tripleExprLabel</a> <span class=math>tl</span> that belongs to <code class=function>tripleExprRefClosure(<span class=math>te</span>)</code>. </li> </ul> <p> The shape expression s1 has a reference to the shape label l2 if </p> <ul> <li>there is a shape sh in atomicShapes(s1) and</li> <li>there is a triple constraint tc in atomicTripleConstraints(sh) and </li> <li>tc.valueExpr is present and </li> <li>tc.valueExpr contains a shape reference to l2.</li> </ul> <p class=leadup> The reference from <span class=math>s1</span> to <span class=math>l2</span> is a <dfn>negated reference</dfn> if the reference to l2 appears under an odd number of ShapeNot in tc.valueExpr. </p> <p> The <dfn>hierarchy and dependency graph</dfn> of a schema is the graph whose nodes are the shape labels that appear in the shape declarations of the schema, and that has an edge from l1 to l2 if: <ul> <li>the definition of l1 has a reference to l2, or</li> <li>there is an edge from l1 to l2 in the hieararchy graph, or</li> <li>there is an edge from l2 to l1 in the hieararchy graph.</li> </ul> The edge from l1 to l2 is negative if the definition of l1 has a negative reference to l2, otherwise the edge is positive. <p></p> <div class=example> <h3 class=followup>Examples with <span class=jobjref><a>ShapeNot</a></span></h3> <p> This negated self-reference violates the negation requirement. </p> <div class="repchoice error"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#S", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p", "valueExpr": { "type": <span class=lookit>"ShapeNot"</span>, "shapeExpr": "http://schema.example/#S" } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:S {ex:p <span class=lookit>NOT</span> @ex:S} &nbsp; </pre> </div> <p> This indirect self-reference does not violate the negation requirement. </p> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#US", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#Up", "valueExpr": { "type": "ShapeNot", "shapeExpr": "http://schema.example/#UT" } } }, { "id": "http://schema.example/#UT", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#Uq", "valueExpr": "http://schema.example/#US" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:US {ex:Up NOT @ex:UT} ex:UT { ex:Uq @ex:US} &nbsp; </pre> </div> <p> This negated, indirect self-reference violates the negation requirement. </p> <div class="repchoice error"> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id" : "http://schema.example/#S", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#p", "valueExpr" : { "type" : "ShapeNot", "shapeExpr" : "http://schema.example/#T" } } } , { "id" : "http://schema.example/#T", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#q", "valueExpr" : "http://schema.example/#S" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:S { ex:p NOT @ex:T } ex:T { ex:q @ex:S } &nbsp; </pre> </div> <p> This is a direct, negated self-reference of the shape with id ex:T and violates the negation requirement. </p> <div class="repchoice error"> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [ { "id" : "http://schema.example/#T", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#p", "valueExpr" : "http://schema.example/#S" } } , { "id" : "http://schema.example/#S", "type" : "ShapeAnd", "shapeExprs" : [ { "type" : "ShapeNot", "shapeExpr" : "http://schema.example/#T" }, "http://schema.example/#U" ] }, { "id" : "http://schema.example/#U", "type" : "Shape" } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:T { ex:p @ex:S } ex:S (NOT @ex:T) AND @ex:U ex:U . &nbsp; </pre> </div> <p> This doubly-negated self-reference of ex:T does not violate the negation requirement. </p> <div class=repchoice> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [{ "id" : "http://schema.example/#T", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#p", "valueExpr" : "http://schema.example/#S" } } , { "id" : "http://schema.example/#S", "type" : "ShapeNot", "shapeExpr" : { "type" : "ShapeAnd", "shapeExprs" : [ { "type" : "ShapeNot", "shapeExpr" : "http://schema.example/#T" }, "http://schema.example/#U" ] } }, { "id" : "http://schema.example/#U", "type" : "Shape" } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:T { ex:p @ex:S } ex:S NOT ( (NOT @ex:T) AND @ex:U ) ex:U . &nbsp; </pre> </div> <p> There is a cycle of negated references between the shape that defines ex:T and the shape that defines ex:U, so the negation requirement is violated. </p> <div class="repchoice error"> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [{ "id" : "http://schema.example/#T", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#p", "valueExpr" : { "type" : "ShapeNot", "shapeExpr" : "http://schema.example/#S" } } } , { "id" : "http://schema.example/#U", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#q", "valueExpr" : "http://schema.example/#S" } }, { "id" : "http://schema.example/#S", "type" : "ShapeAnd", "shapeExprs" : [ { "type" : "ShapeNot", "shapeExpr" : "http://schema.example/#T" }, "http://schema.example/#U" ] } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:T { ex:p NOT @ex:S } ex:U { ex:q @ex:S } ex:S (NOT @ex:T) AND @ex:U &nbsp; </pre> </div> <p> This satisfies the negation requirement, as ex:U does not refer to ex:T (compared to the previous example). </p> <div class=repchoice> <pre class="json nohighlight">{"type" : "Schema", "shapes" : [{ "id" : "http://schema.example/#T", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#p", "valueExpr" : { "type" : "ShapeNot", "shapeExpr" : "http://schema.example/#S" } } } , { "id" : "http://schema.example/#U", "type" : "Shape", "expression" : { "type" : "TripleConstraint", "predicate" : "http://schema.example/#q" } }, { "id" : "http://schema.example/#S", "type" : "ShapeAnd", "shapeExprs" : [ { "type" : "ShapeNot", "shapeExpr" : "http://schema.example/#T" }, "http://schema.example/#U" ] } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:T { ex:p NOT @ex:S } ex:U { ex:q . } ex:S (NOT @ex:T) AND @ex:U &nbsp; </pre> </div> </div> <div class=example> <h3 class=followup>Examples with <span class=jobjref><a>Shape</a></span><span class=param>.extra</span> predicate</h3> <p> This self-reference on a <a>predicate</a> designated as <span class=param>extra</span> violates the negation requirement: </p> <div class="repchoice error"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#S", "type": "Shape", "extra": [ <span class=lookit>"http://schema.example/#p"</span> ], "expression": { "type": "TripleConstraint", "predicate": <span class=lookit>"http://schema.example/#p"</span>, "valueExpr": "http://schema.example/#S" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:S EXTRA ex:p { ex:p @ex:S } &nbsp; </pre> </div> <p> The same shape with a negated self-reference still violates the negation requirement because the reference occurs with a <span class=jobjref><a>ShapeNot</a></span>: </p> <div class="repchoice error"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#S", "type": "Shape", "extra": [ "http://schema.example/#p" ], "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p", "valueExpr": { "type": <span class=lookit>"ShapeNot"</span>, "shapeExpr": "http://schema.example/#S" } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:S EXTRA ex:p { ex:p NOT @ex:S } &nbsp; </pre> </div> </div> </section> <section id=acyclic-extension-requirement> <h4>Acyclic Extension Requirement</h4> <p>The <a>extension hierarchy graph</a> must be acyclic.</p> </section> <section id=extension-coherence-requirement> <h4>Extension Coherence</h4> <p> A <a>shapeDecl</a> <span class=var>D</span> with label <span class=var>L</span> and <span class=math>D.shapeExpr</span> <span class=var>se</span> is called <dfn>extendable</dfn> if it satisfies all of: </p> <ul> <li>it is of the form either `s` or <span class=math>ShapeAnd(s, se)</span>, where `s` is a Shape and <span class=math>se</span> is a shapeExpr. In this case we denote `s` as `<dfn>mainShape</dfn>(L)` and <span class=math>se</span> as `constraint(L)`:</li> <li>`def(L')` is an extendable shape expression for every `L'` in `s.extends` (note that this condition is trivially met when `s.extends` is empty),</li> <li>the set `predicates(<span class=var>se</span>)` is included the union of the sets `predicates(<a>mainShape</a>(L'))` for all shape expression names `L'` that belong to `<a>parentShapeLabels</a>(L)`. </li> </ul> <p> <b>Schema requirement</b> EXTENDS appears only in extendable shape expressions. That is, for every Shape `s` that appears in the schema, if `s.extends` is non empty and for every shapeExpr <span class=math>se</span> in the schema, if <span class=var>s</span> belongs to <span class=math><a>nestedShapes</a>(<span class=var>se</span>)</span>, then <span class=math>se</span> is an extendable shape expression. </p> </section> </section> <section id=semantic-actions> <h3>Semantic Actions</h3> <p> Semantic actions serve as an extension point for Shape Expressions. They appear in lists in <span class=jobjref><a>Schema</a></span>'s <span class=param>startActs</span> and <span class=jobjref><a>Shape</a></span>, <span class=jobjref><a>OneOf</a></span>, <span class=jobjref><a>EachOf</a></span> and <span class=jobjref><a>TripleConstraint</a></span>'s <span class=param>semActs</span>. </p> <p> A semantic action is a tuple of an identifier and some optional code: </p> <div class=shexjTable> <table class=shexj> <tbody><tr class=obj> <th class=obj><dfn>SemAct</dfn></th> <td>{</td> <td><span class=param>name</span>:<a>IRIREF</a> <span class=param>code</span>:<a>STRING</a>? }</td> </tr> </table> </div> <section id=semantic-actions-semantics> <h4>Semantics</h4> <p> The evaluation <span id=semActsSatisfied class=label>semActsSatisfied</span> on a list of <span class=jobjref><a>SemAct</a></span>s returns success or failure. The evaluation of an individual <span class=jobjref><a>SemAct</a></span> is implementation-dependent. </p> </section> <section id=semantic-actions-uses> <h4>Use - informative</h4> <p> A practical evaluation of a <span class=jobjref><a>SemAct</a></span> will provide access to some context. For instance, the <a href=http://shex.io/extensions/Test/#def>http://shex.io/extensions/Test/</a> extension requires access to the subject, <a>predicate</a> and object of a triple matching a <span class=jobjref><a>TripleConstraint</a></span>. These are used in a <code>print</code> function. </p> <div class=example> <div class="example-title marker"><span>Semantic Actions example 1</span></div> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#S1", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#p1", "min": 1, "max": -1, "semActs": [ { "type": "SemAct", "code": " print(s) ", "name": "http://shex.io/extensions/Test/" }, { "type": "SemAct", "code": " print(o) ", "name": "http://shex.io/extensions/Test/" } ] } } ] } </pre> <pre class="shexc nohighlight"> &nbsp; ex:S1 { ex:p1 .+ %Test:{ print(s) %} %Test:{ print(o) %} } &nbsp; </pre> </div> <pre class=data>&lt;http://a.example/n1&gt; &lt;http://a.example/p1&gt; &lt;http://a.example/o1&gt; . &lt;http://a.example/n2&gt; &lt;http://a.example/p1&gt; "a", "b" . &lt;http://a.example/n3&gt; &lt;http://a.example/p2&gt; &lt;http://a.example/o2&gt; .</pre> <table> <tbody><tr> <th class=data>node</th> <th class=schema>shape</th> <th>result</th> <th>print arguments</th> </tr> <tr class=pass> <td>&lt;n1&gt;</td> <td>&lt;S1&gt;</td> <td>pass</td> <td>http://a.example/s1<br>http://a.example/o1</td> </tr> <tr class=pass> <td>&lt;n2&gt;</td> <td>&lt;S1&gt;</td> <td>pass</td> <td>http://a.example/s1<br>"a"<br>http://a.example/s1<br>"b"</td> </tr> <tr class=fail> <td>&lt;n3&gt;</td> <td>&lt;S1&gt;</td> <td>fail</td> <td class=noreason></td> </tr> </table> </div> </section> </section> <section id=annotations> <h3>Annotations</h3> <p> Annotations provide a format-independent way to provide additional information about elements in a schema. They appear in lists in <span class=jobjref><a>Shape</a></span>, <span class=jobjref><a>OneOf</a></span>, <span class=jobjref><a>EachOf</a></span> and <span class=jobjref><a>TripleConstraint</a></span>'s <span class=param>annotations</span>. </p> <div class=shexjTable> <table class=shexj> <tbody><tr class=obj> <th class=obj><dfn>Annotation</dfn></th> <td>{</td> <td><span class=param><a>predicate</a></span>:<a>IRIREF</a> <span class=param>object</span>:<a class=nobref>objectValue</a> }</td> </tr> </table> </div> <section id=annotations-semantics> <h4>Semantics - informative</h4> <p> Annotations do not affect whether a node conforms to some shape. Because they are part of the structure of the schema, they can be parsed in one ShEx format and emitted in that format or another. </p> <div class=example> <div class="example-title marker"><span>Annotations example 1</span></div> <div class=repchoice> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#status", "annotations": [ { "type": "Annotation", "predicate": "http://www.w3.org/2000/01/rdf-schema#comment", "object": {"value": "Represents reported software issues."} }, { "type": "Annotation", "predicate": "http://www.w3.org/2000/01/rdf-schema#label", "object": {"value": "software issue"} } ] } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:status . // rdfs:comment "Represents reported software issues." // rdfs:label "software issue" } &nbsp; </pre> </div> </div> </section> </section> <section id=validation-examples> <h3>Validation Examples</h3> <p> The following examples demonstrate proofs for validations in the form of a nested list of invocations of the evaluation functions defined above. </p> <section id=example-simple> <h4>Simple Examples</h4> <div class=example> <span class=exsec>Schema:</span><br> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">nc1</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IntConstraint", "type": "NodeConstraint", "datatype": "http://www.w3.org/2001/XMLSchema#integer" } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IntConstraint xsd:integer &nbsp; </pre> </div> <div style=clear:both></div> <p> Here the shape identified by <code>http://schema.example/#IntConstraint</code> is a <a>shape expression</a> consisting of a single <span class=jobjref><a class=obj>NodeConstraint</a></span>. Per <a href=#shape-expression-semantics>Shape Expression Semantics</a>, "30"^^&lt;http://www.w3.org/2001/XMLSchema#integer&gt; satisfies <span class="lineno schema">IntConstraint</span>. </p> <p> This document uses this nested tree convention to indicate that the dependency of an evaluation on those nested inside it. Nesting is expressed as indentation. Here, the evaluation of <span class="math function">satisfies NodeConstraint ("30"^^xsd:integer, S1, G, m)</span> depends on <span class="math function">satisfies2 NodeConstraint ("30"^^xsd:integer, S1)</span>. </p> <div class=validation><span>Validate</span> <span>"30"^^&lt;http://www.w3.org/2001/XMLSchema#integer&gt;</span> as <span class=hljs-string>IntConstraint</span>:</div> <ul class="trace followup"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("30"^^xsd:integer, <span class="lineno schema">S1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("30"^^xsd:integer, <span class="lineno schema">S1</span>)</li> </ul> </li> </ul> </div> <p class=leadup> Validating a shape requires evaluating it's <a>triple expression</a> as well as the variables and functions <span class=math><a>neigh</a>(G, n)</span>, <span class=math>matched</span>, <span class=math>remainder</span>, <span class=math>outs</span>, <span class=math>matchables</span> and <span class=math>unmatchables</span>:<br> </p> <div class=example> <span class=exsec>Schema:</span><br> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">tc1</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#UserShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#shoeSize" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:UserShape { ex:shoeSize . } &nbsp; </pre> </div> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels><span class="lineno data">t1</span></pre> <div class="inline json"> <pre>BASE &lt;http://a.example/&gt; PREFIX xsd: &lt;http://www.w3.org/2001/XMLSchema#&gt; &lt;Alice&gt; ex:shoeSize "30"^^xsd:integer . </pre> </div> <div style=clear:both class=validation><span>Validate</span> <span class=hljs-name>&lt;Alice&gt;</span> as <span class=hljs-string>http://schema.example/#UserShape</span>:</div> <ul class="trace followup"> <li>G = [<span class="lineno data">t1</span>] <span class=comment>The <a>graph</a> G consists of one <a>triple</a>.</span></li> <li><a href=#satisfies-Shape>satisfies Shape</a> (&lt;Alice&gt;, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, &lt;Alice&gt;) = [<span class="lineno data">t1</span>] <span class=comment>/\* The neighborhood around &lt;Alice&gt; consists of one triple. \*/</span></li> <li>matched = [<span class="lineno data">t1</span>] <span class=comment>/\* That triple is matched in the nested evaluation. \*/</span></li> <li>remainder = %ÿ <span class=comment>/\* The remainder is the empty set. \*/</span> </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m)</li> <li>outs = [<span class="lineno data">t1</span>] <span class=comment>/\* There is one arc out. \*/</span></li> <li>matchables = %ÿ <span class=comment>/\* There are no remaining arcs out of &lt;Alice&gt; with <a data-lt=predicate>predicates</a> appearing in tc1. \*/</span></li> <li>unmatchables = %ÿ <span class=comment>/\* There are no other arcs out of &lt;Alice&gt;. \*/</span></li> <li><span class=parameter>closed</span> is false <span class=comment>/\* The <span class=jobjref><a>Shape</a></span>'s closed paramater has a value of false. \*/</span></li> </ul> </li> </ul> </div> <p class=leadup> It is quite common that Shapes will constrain their nested TripleConstraints with NodeConstraints. Here is an example including that, extra triples and a closed shape: </p> <div class=example> <span class=exsec>Schema:</span><br> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">tc1</span> <span class="lineno schema">nc1</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#UserShape", "type": "Shape", "extra": ["http://www.w3.org/1999/02/22-rdf-syntax-ns#type"], "expression": { "type": "TripleConstraint", "predicate": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type", "valueExpr": { "type": "NodeConstraint", "values": ["http://schema.example/#Teacher"] } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:UserShape EXTRA a { a [ex:Teacher] } &nbsp; </pre> </div> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels><span class="lineno data">t1</span> <span class="lineno data">t2</span> <span class="lineno data">t3</span> <span class="lineno data">t4</span> <span class="lineno data">t5</span></pre> <div class="inline json"> <pre>BASE &lt;http://a.example/&gt; PREFIX xsd: &lt;http://www.w3.org/2001/XMLSchema#&gt; &lt;Alice&gt; ex:shoeSize "30"^^xsd:integer . &lt;Alice&gt; a ex:Teacher . &lt;Alice&gt; a ex:Person . &lt;SomeHat&gt; ex:owner &lt;Alice&gt; . &lt;TheMoon&gt; ex:madeOf &lt;GreenCheese&gt; . </pre> </div> <div style=clear:both class=validation><span>Validate</span> <span class=hljs-name>&lt;Alice&gt;</span> as <span class=hljs-string>http://schema.example/#UserShape</span>:</div> <ul class="trace followup"> <li>G = [<span class="lineno data">t1,t2,t3,t4,t5</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (&lt;Alice&gt;, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, &lt;Alice&gt;) = [<span class="lineno data">t1,t2,t3,t4</span>], matched = [<span class="lineno data">t2</span>], remainder = [<span class="lineno data">t1,t3</span>]</li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t2</span>], <span class="lineno schema">tc1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (ex:Teacher, <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (ex:Teacher, <span class="lineno schema">nc1</span>)</li> </ul> </li> </ul> </li> <li>outs = [<span class="lineno data">t1,t2,t3</span>]</li> <li>matchables = [<span class="lineno data">t3</span>], unmatchables = [<span class="lineno data">t1</span>], <span class=parameter>closed</span> is false </li> </ul> </li> </ul> <p> The non-empty matchables is permitted because the triple <code>t3</code> has a <a>predicate</a> which appears in the "extra" list: <span class=json>["http://schema.example/#Teacher"]</span>. </p> </div> </section> <section id=example-disjunction> <h4>Disjunction Example</h4> <div class=example> <div class=exsec>Schema:</div><br> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">te1</span> <span class="lineno schema">tc1</span> <span class="lineno schema">nc1</span> <span class="lineno schema">te2</span> <span class="lineno schema">tc2</span> <span class="lineno schema">nc2</span> <span class="lineno schema">tc3</span> <span class="lineno schema">nc3</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#UserShape", "type": "Shape", "expression": {"type": "OneOf", "expressions": [ { "type": "TripleConstraint", "predicate": "http://xmlns.com/foaf/0.1/name", "valueExpr": { "type": "NodeConstraint", "nodeKind": "literal" } }, { "type": "EachOf", "expressions": [ { "type": "TripleConstraint", "min": 1, "max": -1 , "predicate": "http://xmlns.com/foaf/0.1/givenName", "valueExpr": { "type": "NodeConstraint", "nodeKind": "literal" } }, { "type": "TripleConstraint", "predicate": "http://xmlns.com/foaf/0.1/familyName", "valueExpr": { "type": "NodeConstraint", "nodeKind": "literal" } } ] } ] } } ] } </pre> <pre class="shexc nohighlight"> &nbsp; ex:UserShape { ( <span class=comment># extra ()s to clarify alignment with <a>ShExJ</a></span> foaf:name LITERAL | ( <span class=comment># extra ()s to clarify alignment with <a>ShExJ</a></span> foaf:givenName LITERAL+ ; foaf:familyName LITERAL ) ) } </pre> </div> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels> <span class="lineno data">t1</span> <span class="lineno data">t2</span> <span class="lineno data">t3</span> <span class="lineno data">t4</span> <span class="lineno data">t5</span> <span class="lineno data">t6</span></pre> <div class="inline json"> <pre>BASE &lt;http://a.example/&gt; PREFIX foaf: &lt;http://xmlns.com/foaf/0.1/&gt; &lt;Alice&gt; foaf:givenName "Alice" . &lt;Alice&gt; foaf:givenName "Malsenior" . &lt;Alice&gt; foaf:familyName "Walker" . &lt;Alice&gt; foaf:mbox &lt;mailto:alice@example.com&gt; . &lt;Bob&gt; foaf:knows &lt;Alice&gt; . &lt;Bob&gt; foaf:mbox &lt;mailto:bob@example.com&gt; . </pre> </div> <div style=clear:both></div> <p class=leadup>Per <a href=#shape-expression-semantics>Shape Expression Semantics</a>, &lt;Alice&gt; satisfies <span class=lineno>S1</span> with the simple <a>ShapeMap</a></p> <pre class=line-labels>m:</pre> <div class="inline json"> <pre>{ "http://a.example/Alice": "http://a.example/UserShape }</pre> </div> <div style=clear:both></div> <p class="leadup followup">as seen in this validation.</p> <div class=validation><span>Validate</span> <span class=hljs-name>&lt;Alice&gt;</span> as <span class=hljs-string>http://schema.example/#UserShape</span>:</div> <ul class="trace followup"> <li>G = [<span class="lineno data">t1,t2,t3,t4,t5,t6</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (&lt;Alice&gt;, <span class=lineno>S1</span>, G, m) <ul> <li><a>neigh</a>(G, &lt;Alice&gt;) = [<span class="lineno data">t1,t2,t3,t4,t5</span>], matched = [<span class="lineno data">t1,t2,t3</span>], remainder = [<span class="lineno data">t4,t5</span>]</li> <li><a href=#matches-OneOf>matches OneOf</a> ([<span class="lineno data">t1,t2,t3</span>], <span class="lineno schema">te1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-EachOf>matches EachOf</a> ([<span class="lineno data">t1,t2,t3</span>], <span class="lineno schema">te2</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data">t1,t2</span>], <span class="lineno schema">tc2</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc2</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("Alice", <span class="lineno schema">nc2</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("Alice", <span class="lineno schema">nc2</span>)</li> </ul> </li> </ul> </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t2</span>], <span class="lineno schema">tc2</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("Malsenior", <span class="lineno schema">nc2</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("Malsenior", <span class="lineno schema">nc2</span>)</li> </ul> </li> </ul> </li> </ul> </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t3</span>], <span class="lineno schema">tc3</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("Walker", <span class="lineno schema">nc3</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("Walker", <span class="lineno schema">nc3</span>)</li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> <li>outs = [<span class="lineno data">t4</span>] <span class=comment>/\* <span class="lineno data">t5</span> is in <span class=math><a>ArcsIn</a>(G, &lt;Alice&gt;)&gt;</span>, <span class="lineno data">t6</span> is not in <span class=math><a>neigh</a>(G, &lt;Alice&gt;)&gt;</span>. \*/</span></li> <li>matchables = %ÿ, unmatchables = [<span class="lineno data">t5</span>], <span class=parameter>closed</span> is false</li> </ul> </li> </ul> <p class=leadup> Replacing triples 1-3 with a single foaf:name property will also satisfy the schema. </p> <div class=exsec>Data:</div> <pre class=line-labels> <span class="lineno data">t4</span> <span class="lineno data">t5</span> <span class="lineno data">t6</span> <span class="lineno data">t7</span></pre> <div class="inline json"> <pre>BASE &lt;http://a.example/&gt; PREFIX foaf: &lt;http://xmlns.com/foaf/0.1/&gt; &lt;Alice&gt; foaf:mbox &lt;mailto:alice@example.com&gt; . &lt;Bob&gt; foaf:knows &lt;Alice&gt; . &lt;Bob&gt; foaf:mbox &lt;mailto:bob@example.com&gt; . &lt;Alice&gt; foaf:name "Alice Malsenior Walker" . </pre> </div> <div style=clear:both class=validation><span>Validate</span> <span class=hljs-name>&lt;Alice&gt;</span> as <span class=hljs-string>http://schema.example/#UserShape</span>:</div> <ul class="trace followup"> <li>G = [<span class="lineno data">t4,t5,t6,t7</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (&lt;Alice&gt;, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, &lt;Alice&gt;) = [<span class="lineno data">t4,t5,t7</span>], matched = [<span class="lineno data">t7</span>], remainder = [<span class="lineno data">t4,t5</span>]</li> <li><a href=#matches-OneOf>matches OneOf</a> ([<span class="lineno data">t7</span>], <span class="lineno schema">te1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t7</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("Walker", <span class="lineno schema">nc3</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("Walker", <span class="lineno schema">nc3</span>)</li> </ul> </li> </ul> </li> </ul> </li> <li>outs = [<span class="lineno data">t4</span>]</li> <li>matchables = %ÿ, unmatchables = [<span class="lineno data">t5</span>], <span class=parameter>closed</span> is false</li> </ul> </li> </ul> <p class=leadup> Any mixure of <code>foaf:name</code> with <code>foaf:givenName</code> or <code>foaf:familyName</code> will fail to satisfy the schema as there will be a matchable triple <span class="lineno data">t3</span> that is not used in the <a>triple expression</a> <span class="lineno schema">te1</span>. </p> <div class=exsec>Data:</div> <pre class=line-labels> <span class="lineno data">t3</span> <span class="lineno data">t4</span> <span class="lineno data">t5</span> <span class="lineno data">t6</span> <span class="lineno data">t7</span></pre> <div class="inline json"> <pre>BASE &lt;http://a.example/&gt; PREFIX foaf: &lt;http://xmlns.com/foaf/0.1/&gt; <span class=fail>&lt;Alice&gt; foaf:familyName "Walker" .</span> &lt;Alice&gt; foaf:mbox &lt;mailto:alice@example.com&gt; . &lt;Bob&gt; foaf:knows &lt;Alice&gt; . &lt;Bob&gt; foaf:mbox &lt;mailto:bob@example.com&gt; . &lt;Alice&gt; foaf:name "Alice Malsenior Walker" . </pre> </div> <div style=clear:both class=validation><span>Validate</span> <span class=hljs-name>&lt;Alice&gt;</span> as <span class=hljs-string>http://schema.example/#UserShape</span>:</div> <ul class="trace followup"> <li>G = [<span class="lineno data">t4,t5,t6,t7</span>]</li> <li><span class=fail><a href=#satisfies-Shape>satisfies Shape</a> (&lt;Alice&gt;, <span class="lineno schema">S1</span>, G, m)</span> <ul> <li><a>neigh</a>(G, &lt;Alice&gt;) = [<span class="lineno data">t4,t5,t7</span>], matched = [<span class="lineno data">t7</span>], remainder = [<span class="lineno data">t4,t5</span>]</li> <li><a href=#matches-OneOf>matches OneOf</a> ([<span class="lineno data">t7</span>], <span class="lineno schema">te1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t7</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("Walker", <span class="lineno schema">nc3</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("Walker", <span class="lineno schema">nc3</span>)</li> </ul> </li> </ul> </li> </ul> </li> <li>outs = [<span class="lineno data">t4</span>]</li> <li><span class=fail>matchables = [<span class="lineno data">t3</span>]</span>, unmatchables = [<span class="lineno data">t5</span>], <span class=parameter>closed</span> is false</li> </ul> </li> </ul> <p class=leadup> Adding a <code>foaf:familyName</code> to <span class=math><span class="lineno schema">S1</span></span>'s <span class=param>extra</span> would allow this <a>graph</a> to satisfy the schema. </p> <pre class=line-labels><span class="lineno schema">S1</span> </pre> <div class="inline json"> <pre>{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#UserShape", "type": "Shape", "extra": ["http://xmlns.com/foaf/0.1/familyName"] ÔÇª } ] }</pre> </div> <div style=clear:both></div> <p class=leadup> Closing <span class=math><span class="lineno schema">S1</span></span> would also cause a validation failure if <span class=math>unmatchables</span> were not empty: </p> <pre class=line-labels><span class="lineno schema">S1</span> </pre> <div class="inline json"> <pre>{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#UserShape", "type": "Shape", "closed": true ÔÇª } ] }</pre> </div> <div style=clear:both></div> <ul class="trace followup"> <li>G = [<span class="lineno data">t4,t5,t6,t7</span>]</li> <li><span class=fail><a href=#satisfies-Shape>satisfies Shape</a> (&lt;Alice&gt;, <span class="lineno schema">S1</span>, G, m)</span> <ul> <li>ÔÇª</li> <li><span class=fail>unmatchables = [<span class="lineno data">t5</span>], <span class=parameter>closed</span> is true</span></li> </ul> </li> </ul> </div> </section> <section id=example-dependent-shape> <h4>Dependent Shape Example</h4> <div class=example> <div class=exsec>Schema:</div> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">tc1</span> <span class="lineno schema">nc1</span> <span class="lineno schema">S2</span> <span class="lineno schema">tc2</span> <span class="lineno schema">nc2</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#reproducedBy", "valueExpr": "http://schema.example/#TesterShape" } }, { "id": "http://schema.example/#TesterShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#role", "valueExpr": { "type": "NodeConstraint", "values": [ "http://schema.example/#testingRole" ] } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:reproducedBy @ex:TesterShape } ex:TesterShape { ex:role [ex:testingRole] } &nbsp; </pre> </div> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels><span class="lineno data">t1</span> <span class="lineno data">t2</span></pre> <div class="inline json"> <pre>PREFIX ex: &lt;http://schema.example/#&gt; PREFIX inst: &lt;http://inst.example/&gt; inst:Issue1 ex:reproducedBy inst:Tester2 . inst:Tester2 ex:role ex:testingRole .</pre> </div> <div style=clear:both></div> <p class=leadup>inst:Issue1 satisfies <span class="lineno schema">S1</span> with the <a>ShapeMap</a></p> <pre class=line-labels>m:</pre> <div class="inline json"> <pre>{ "http://inst.example/Issue1": "http://schema.example/#IssueShape", "http://inst.example/Tester2": "http://schema.example/#TesterShape", "http://inst.example/Testgrammer23": "http://schema.example/#ProgrammerShape" }</pre> </div> <div style=clear:both class=validation><span>Validate</span> <span>inst:Issue1</span> as <span class=hljs-string>http://schema.example/#IssueShape</span>:</div> <p class="leadup followup">as seen in this evaluation:</p> <ul class="trace followup"> <li>G = [<span class="lineno data">t1</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (inst:Issue1, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, inst:Issue1) = [<span class="lineno data">t1,t2</span>], matched = [<span class="lineno data">t1,t2</span>], remainder = %ÿ</li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (inst:Tester2, <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (inst:Tester2, <span class="lineno schema">nc1</span>) <ul> <li><a href=#satisfies-Shape>satisfies Shape</a> (inst:Tester2. <span class="lineno schema">S2</span>, G, m) <ul> <li><a>neigh</a>(G, inst:Tester2) = [<span class="lineno data">t2</span>], matched = [<span class="lineno data">t2</span>], remainder = %ÿ </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t2</span>], <span class="lineno schema">tc2</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (ex:testingRole, <span class="lineno schema">nc2</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (ex:testingRole, <span class="lineno schema">nc2</span>) </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, closed is false </li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, closed is false</li> </ul> </li> </ul> <div style=clear:both></div> </div> </section> <section id=example-recursion> <h4>Recursion Example</h4> <div class=example> <div class=exsec>Schema:</div> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">tc1</span> <span class="lineno schema">nc1</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "TripleConstraint", "min": 0, "max": -1, "predicate": "http://schema.example/#related", "valueExpr": "http://schema.example/#IssueShape" } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:related @ex:IssueShape\* }</pre> </div> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels><span class="lineno data">t1</span> <span class="lineno data">t2</span> <span class="lineno data">t3</span></pre> <div class="inline json"> <pre>PREFIX ex: &lt;http://schema.example/#&gt; PREFIX inst: &lt;http://inst.example/&gt; inst:Issue1 ex:related inst:Issue2 . inst:Issue2 ex:related inst:Issue3 . inst:Issue3 ex:related inst:Issue1 .</pre> </div> <div style=clear:both></div> <p class=leadup>inst:Issue1 satisfies <span class="lineno schema">S1</span> with the <a>ShapeMap</a></p> <pre class=line-labels>m:</pre> <div class="inline json"> <pre>{ "http://inst.example/Issue1": "http://schema.example/#IssueShape", "http://inst.example/Issue2": "http://schema.example/#IssueShape", "http://inst.example/Issue3": "http://schema.example/#IssueShape" }</pre> </div> <div style=clear:both class=validation><span>Validate</span> <span>inst:Issue1</span> as <span class=hljs-string>http://schema.example/#IssueShape</span>:</div> <p class="leadup followup">as seen in this evaluation:</p> <ul class="trace followup"> <li>G = [<span class="lineno data">t1,t2,t3</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (inst:Issue1, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, inst:Issue1) = [<span class="lineno data">t1</span>], matched = [<span class="lineno data">t1</span>], remainder = %ÿ</li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (inst:Issue2, <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (inst:Issue2, <span class="lineno schema">nc1</span>) <ul> <li><a href=#satisfies-Shape>satisfies Shape</a> (inst:Issue2. <span class="lineno schema">S2</span>, G, m) <ul> <li><a>neigh</a>(G, inst:Issue2) = [<span class="lineno data">t3</span>], matched = [<span class="lineno data">t3</span>], remainder = %ÿ </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t3</span>], <span class="lineno schema">tc3</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (inst:Issue3, <span class="lineno schema">nc3</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (inst:Issue3, <span class="lineno schema">nc3</span>) <ul> <li><a href=#satisfies-Shape>satisfies Shape</a> (inst:Issue3. <span class="lineno schema">S2</span>, G, m) <ul> <li><a>neigh</a>(G, inst:Issue3) = [<span class="lineno data">t3</span>], matched = [<span class="lineno data">t3</span>], remainder = %ÿ</li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t3</span>], <span class="lineno schema">tc3</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (inst:Issue1, <span class="lineno schema">nc3</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (inst:Issue1, <span class="lineno schema">nc3</span>)<br> This is known to be true or the initial typing would not be satisfied. </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, closed is false</li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, closed is false </li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, closed is false</li> </ul> </li> </ul> <div style=clear:both></div> </div> </section> <section id=example-simple-repeated-property> <h4>Simple Repeated Property Examples</h4> <div class=example> <div class=exsec>Schema:</div> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">te1</span> <span class="lineno schema">tc1</span> <span class="lineno schema">nc1</span> <span class="lineno schema">tc2</span> <span class="lineno schema">nc2</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#TestResultsShape", "type": "Shape", "expression": { "type": "EachOf", "expressions": [ { "type": "TripleConstraint", "min": 1, "max": -1, "predicate": "http://schema.example/#val", "valueExpr": { "type": "NodeConstraint", "values": [ {"value": "a"}, {"value": "b"}, {"value": "c"} ] } }, { "type": "TripleConstraint", "min": 1, "max": -1, "predicate": "http://schema.example/#val", "valueExpr": { "type": "NodeConstraint", "values": [ {"value": "b"}, {"value": "c"}, {"value": "d"} ] } } ] } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; &lt;http://schema.example/#TestResultsShape&gt; { &lt;http://schema.example/#val&gt; ["a" "b" "c"]+ ; &lt;http://schema.example/#val&gt; ["b" "c" "d"]+ } </pre> </div> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels> <span class="lineno data">t1</span> <span class="lineno data">t2</span> <span class="lineno data">t3</span> <span class="lineno data">t4</span></pre> <div class="inline json"> <pre>BASE &lt;http://a.example/&gt; PREFIX ex: &lt;http://schema.example/#&gt; &lt;s&gt; ex:val "a" . &lt;s&gt; ex:val "b" . &lt;s&gt; ex:val "c" . &lt;s&gt; ex:val "d" .</pre> </div> <div style=clear:both></div> <p class=leadup>&lt;s&gt; satisfies <span class="lineno schema">S1</span> with:</p> <pre class=line-labels>m:</pre> <div class="inline json"> <pre>{ "http://a.example/s": "http://a.example/S1" }</pre> </div> <div style=clear:both class=validation><span>Validate</span> <span class=hljs-name>&lt;s&gt;</span> as <span class=hljs-string>http://schema.example/#TestResultShape</span>:</div> <p class="leadup followup">If <span class=math><span class="lineno schema">tc1</span></span> consumes as many triples as it can, it consumes three and <span class=math><span class="lineno schema">tc2</span></span> consumes one:</p> <ul class="trace followup"> <li>G = [<span class="lineno data">t1,t2,t3,t4</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (&lt;s&gt;, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, &lt;s&gt;) = [<span class="lineno data">t1,t2,t3,t4</span>], matched = [<span class="lineno data">t1,t2,t3,t4</span>], remainder = %ÿ</li> <li><a href=#matches-EachOf>matches EachOf</a> ([<span class="lineno data">t1,t2,t3,t4</span>], <span class="lineno schema">te1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data">t1,t2,t3</span>], <span class="lineno schema">tc1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("a", <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("a", <span class="lineno schema">nc1</span>)</li> </ul> </li> </ul> </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t2</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("b", <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("b", <span class="lineno schema">nc1</span>)</li> </ul> </li> </ul> </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t3</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("c", <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("c", <span class="lineno schema">nc1</span>)</li> </ul> </li> </ul> </li> </ul> </li> <li><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data">t4</span>], <span class="lineno schema">tc2</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t4</span>], <span class="lineno schema">tc2</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("d", <span class="lineno schema">nc2</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("d", <span class="lineno schema">nc2</span>)</li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, <span class=parameter>closed</span> is false </li> </ul> </li> </ul> <div style=clear:both></div> <p class=leadup>If we eliminate <span class=math><span class="lineno data">t4</span></span>, either <span class=math><span class="lineno data">t2</span></span> or <span class=math><span class="lineno data">t3</span></span> must be allocated to <span class=math><span class="lineno schema">tc2</span></span>:</p> <ul class="trace followup"> <li>G = [<span class="lineno data">t1,t2,t3</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (&lt;Alice&gt;, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, &lt;Alice&gt;) = [<span class="lineno data">t1,t2,t3</span>], matched = [<span class="lineno data">t1,t2,t3</span>], remainder = %ÿ</li> <li><a href=#matches-EachOf>matches EachOf</a> ([<span class="lineno data">t1,t2,t3</span>], <span class="lineno schema">te1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data">t1,t2</span>], <span class="lineno schema">tc1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("a", <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("a", <span class="lineno schema">nc1</span>)</li> </ul> </li> </ul> </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t2</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("b", <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("b", <span class="lineno schema">nc1</span>)</li> </ul> </li> </ul> </li> </ul> </li> <li><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data">t3</span>], <span class="lineno schema">tc2</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t3</span>], <span class="lineno schema">tc2</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("d", <span class="lineno schema">nc2</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("d", <span class="lineno schema">nc2</span>)</li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, <span class=parameter>closed</span> is false </li> </ul> </li> </ul> <div style=clear:both></div> </div> </section> <section id=example-repeated-property-with-dependent-shapes> <h4>Repeated Property With Dependent Shapes Example</h4> <div class=example> <div class=exsec>Schema:</div> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">te1</span> <span class="lineno schema">tc1</span> <span class="lineno schema">nc1</span> <span class="lineno schema">tc2</span> <span class="lineno schema">nc2</span> <span class="lineno schema">S2</span> <span class="lineno schema">tc3</span> <span class="lineno schema">nc3</span> <span class="lineno schema">S3</span> <span class="lineno schema">tc4</span> <span class="lineno schema">nc4</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#IssueShape", "type": "Shape", "expression": { "type": "EachOf", "expressions": [ { "type": "TripleConstraint", "predicate": "http://schema.example/#reproducedBy", "valueExpr": "http://schema.example/#TesterShape" }, { "type": "TripleConstraint", "predicate": "http://schema.example/#reproducedBy", "valueExpr": "http://schema.example/#ProgrammerShape" } ] } }, { "id": "http://schema.example/#TesterShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#role", "valueExpr": { "type": "NodeConstraint", "values": [ "http://schema.example/#testingRole" ] } } }, { "id": "http://schema.example/#ProgrammerShape", "type": "Shape", "expression": { "type": "TripleConstraint", "predicate": "http://schema.example/#department", "valueExpr": { "type": "NodeConstraint", "values": [ "http://schema.example/#ProgrammingDepartment" ] } } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; ex:IssueShape { ex:reproducedBy @ex:TesterShape; ex:reproducedBy @ex:ProgrammerShape } ex:TesterShape { ex:role [ex:testingRole] } ex:ProgrammerShape { ex:department [ex:ProgrammingDepartment] } &nbsp;</pre> </div> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels> <span class="lineno data">t1</span> <span class="lineno data">t2</span> <span class="lineno data">t3</span> <span class="lineno data">t4</span> <span class="lineno data">t5</span></pre> <div class="inline json"> <pre>PREFIX ex: &lt;http://schema.example/#&gt; PREFIX inst: &lt;http://inst.example/&gt; inst:Issue1 ex:reproducedBy inst:Tester2 ; ex:reproducedBy inst:Testgrammer23 . inst:Tester2 ex:role ex:testingRole . inst:Testgrammer23 ex:role ex:testingRole ; ex:department ex:ProgrammingDepartment .</pre> </div> <div style=clear:both></div> <p class=leadup>inst:Issue1 satisfies <span class="lineno schema">S1</span> with the <a>ShapeMap</a></p> <pre class=line-labels>m:</pre> <div class="inline json"> <pre>{ "http://inst.example/Issue1": "http://schema.example/#IssueShape", "http://inst.example/Tester2": "http://schema.example/#TesterShape", "http://inst.example/Testgrammer23": "http://schema.example/#ProgrammerShape" }</pre> </div> <div style=clear:both class=validation><span>Validate</span> <span>inst:Issue1</span> as <span class=hljs-string>http://schema.example/#IssueShape</span>:</div> <p class="leadup followup">as seen in this evaluation:</p> <ul class="trace followup"> <li>G = [<span class="lineno data">t1,t2,t3,t4,t5</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (inst:Issue1, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, inst:Issue1) = [<span class="lineno data">t1,t2</span>], matched = [<span class="lineno data">t1,t2</span>], remainder = %ÿ</li> <li><a href=#matches-EachOf>matches EachOf</a> ([<span class="lineno data">t1,t2</span>], <span class="lineno schema">te1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (inst:Tester2, <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (inst:Tester2, <span class="lineno schema">nc1</span>) <ul> <li><a href=#satisfies-Shape>satisfies Shape</a> (inst:Tester2. <span class="lineno schema">S2</span>, G, m) <ul> <li><a>neigh</a>(G, inst:Tester2) = [<span class="lineno data">t3</span>], matched = [<span class="lineno data">t3</span>], remainder = %ÿ</li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t3</span>], <span class="lineno schema">tc3</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (ex:testingRole, <span class="lineno schema">nc3</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (ex:testingRole, <span class="lineno schema">nc3</span>) </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, closed is false</li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t2</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (inst:Testgrammer23, <span class="lineno schema">nc2</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (inst:Testgrammer23, <span class="lineno schema">nc2</span>) <ul> <li><a href=#satisfies-Shape>satisfies Shape</a> (inst:Testgrammer23. <span class="lineno schema">S3</span>, G, m) <ul> <li><a>neigh</a>(G, inst:Testgrammer23) = [<span class="lineno data">t5</span>], matched = [<span class="lineno data">t5</span>], remainder = %ÿ</li> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t5</span>], <span class="lineno schema">tc3</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> (ex:testingRole, <span class="lineno schema">nc4</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> (ex:testingRole, <span class="lineno schema">nc4</span>) </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, closed is false</li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, closed is false</li> </ul> </li> </ul> <div style=clear:both></div> </div> </section> <section id=example-negation> <h4>Negation Example</h4> <p> Setting the maximum <a>cardinality</a> of a TripleConstraint with <a>predicate</a> <span class=math>p</span> to zero (i.e. <span>"max": 0</span> in <a>ShExJ</a> or <code>{0}</code> or <code>{0, 0}</code> in <a>ShExC</a>) asserts that matching nodes must have no triples with <a>predicate</a> <span class=math>p</span>. </p> <div class=example> <div class=exsec>Schema:</div> <pre class=line-labels><span class="lineno schema">S1</span> <span class="lineno schema">te1</span> <span class="lineno schema">tc1</span> <span class="lineno schema">nc1</span> <span class="lineno schema">tc2</span> </pre> <div class="inline repchoice"> <pre class="json nohighlight">{ "type": "Schema", "shapes": [ { "id": "http://schema.example/#TestResultsShape", "type": "Shape", "expression": { "type": "EachOf", "expressions": [ { "type": "TripleConstraint", "min": 1, "max": -1, "predicate": "http://schema.example/#p1", "valueExpr": { "type": "NodeConstraint", "values": [ {"value": "a"}, {"value": "b"} ] } }, { "type": "TripleConstraint", "min": 1, "max": -1, "predicate": "http://schema.example/#p2", "min": 0, "max": 0 } ] } } ] }</pre> <pre class="shexc nohighlight"> &nbsp; &lt;http://schema.example/#TestResultsShape&gt; { &lt;http://schema.example/#p1&gt; ["a" "b"] + ; &lt;http://schema.example/#p2&gt; . {0} } </pre> </div> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels> <span class="lineno data">t1</span></pre> <div class="inline json"> <pre>BASE &lt;http://a.example/&gt; PREFIX ex: &lt;http://schema.example/#&gt; &lt;s&gt; ex:p1 "a" .</pre> </div> <div style=clear:both></div> <p class=leadup>&lt;s&gt; satisfies <span class="lineno schema">S1</span> with:</p> <pre class=line-labels>m:</pre> <div class="inline json"> <pre>{ "http://a.example/s": "http://a.example/S1" }</pre> </div> <div style=clear:both class=validation><span>Validate</span> <span class=hljs-name>&lt;s&gt;</span> as <span class=hljs-string>http://schema.example/#TestResultShape</span>:</div> <p class="leadup followup">This is trivially satisfied by <span class=math><span class="lineno schema">tc1</span></span> consuming one triple and <span class=math><span class="lineno schema">tc2</span></span> consuming none:</p> <ul class="trace followup"> <li>G = [<span class="lineno data">t1</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (&lt;s&gt;, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, &lt;s&gt;) = [<span class="lineno data">t1</span>], matched = [<span class="lineno data">t1</span>], remainder = %ÿ</li> <li><a href=#matches-EachOf>matches EachOf</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">te1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("a", <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("a", <span class="lineno schema">nc1</span>)</li> </ul> </li> </ul> </li> </ul> </li> <li><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data"></span>], <span class="lineno schema">tc2</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data"></span>], <span class="lineno schema">tc2</span>, m)</li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, <span class=parameter>closed</span> is false </li> </ul> </li> </ul> <div style=clear:both></div> <p class=leadup>If we add a <span class=math><span class="lineno data">t2</span></span> which matches <span class=math><span class="lineno schema">tc2</span></span>:</p> <div style=clear:both class=exsec>Data:</div> <pre class=line-labels> <span class="lineno data">t1</span> <span class="lineno data">t2</span></pre> <div class="inline json error"> <pre>BASE &lt;http://a.example/&gt; PREFIX ex: &lt;http://schema.example/#&gt; &lt;s&gt; ex:p1 "a" . &lt;s&gt; ex:p2 5 .</pre> </div> <div style=clear:both></div> <p> every partition fails, either because matchables is non-empty or because the maximum cardinality on <span class="lineno schema">tc2</span> is exceeded: </p> <ul class="trace followup"> <li>G = [<span class="lineno data">t1</span>]</li> <li><a href=#satisfies-Shape>satisfies Shape</a> (&lt;s&gt;, <span class="lineno schema">S1</span>, G, m) <ul> <li><a>neigh</a>(G, &lt;s&gt;) = [<span class="lineno data">t1</span>], matched = [<span class="lineno data">t1</span>], remainder = %ÿ</li> <li><a href=#matches-EachOf>matches EachOf</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">te1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t1</span>], <span class="lineno schema">tc1</span>, m) <ul> <li><a href=#satisfies-NodeConstraint>satisfies NodeConstraint</a> ("a", <span class="lineno schema">nc1</span>, G, m) <ul> <li><a href=#satisfies2-NodeConstraint>satisfies2 NodeConstraint</a> ("a", <span class="lineno schema">nc1</span>)</li> </ul> </li> </ul> </li> </ul> </li> <li><span class=fail><a href=#matches-cardinality>matches cardinality</a> ([<span class="lineno data">t2</span>], <span class="lineno schema">tc2</span>, m)</span> <ul style="border-left:solid thin brown"> <li><a href=#matches-TripleConstraint>matches TripleConstraint</a> ([<span class="lineno data">t2</span>], <span class="lineno schema">tc2</span>, m)</li> </ul> </li> </ul> </li> <li>outs = %ÿ</li> <li>matchables = %ÿ, unmatchables = %ÿ, <span class=parameter>closed</span> is false </li> </ul> </li> </ul> <div style=clear:both></div> </div> </section> </section> </section> <section id=shexc> <h2>ShEx Compact syntax (ShExC)</h2> <div> <p> The ShEx Compact Syntax expresses ShEx schemas in a compact, human-friendly form. Parsing <dfn>ShExC</dfn> transforms a <a>ShExC</a> document into an equivalent <a>ShExJ</a> structure. This is defined as a BNF which accepts <a>ShExC</a> followed by instructions for tranlating the rules in the BNF production into their corresponding <a>ShExJ</a> objects. For example, "<span style=font-family:monospace>shapeExprDecl returns <span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span></span>" indicates that the result of matching the <code>shapeExprDecl</code> production is the object produced by parsing the <code>shapeExpression</code> production. </p> <p> Semantic actions before the first <span class=prod><a class=grammarRef href=#prod-shapeExprDecl>shape expression declaration</a></span> are <span class=param><a>startActs</a></span>. After the first <span class=prod><a class=grammarRef href=#prod-shapeExprDecl>shape expression declaration</a></span>, semantic actions are associated with the previous declaration. </p> <p> As with Turtle and SPARQL, ShExC offers URL resolution relative to a <dfn data-lt=prefix>base</dfn> per [[RFC3986]] and <dfn>prefixes map</dfn> to provide shorthand ways to write IRI identifiers. </p> <p> <button id=toggleGrammar>Display grammar only</button> </p> <div class=grammarTable> <table style="border:0;border-top:thick solid white"> <tbody class=grammarParagraph> <tr style="border-top:thick solid white;border-left:thin solid white;border-right:thin solid white"> <td colspan=4>Below is the <a>ShExC</a> grammar following the <a data-cite=XML#sec-notation>notation in the XML specification</a>[[!XML]]:</td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-shexDoc>[<span class=prodNo>1</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shexDoc</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-directive>directive</a></span>\* ((<span class=prod><a class=grammarRef href=#prod-notStartAction>notStartAction</a></span> | <span class=prod><a class=grammarRef href=#prod-startActions>startActions</a></span>) <span class=prod><a class=grammarRef href=#prod-statement>statement</a></span>\*)?</code> </td> </tr> </tbody> <tbody class=grammarParagraph> <tr> <td colspan=4 style=padding-top:2ex>followed by the associated <a>ShExJ</a> object(s):</td> </tr> </tbody> <tbody class=prod> <tr class=obj> <td style="border-top:thick solid white;border-bottom:thick solid white"></td> <td><a class=obj>Schema</a></td> <td>{</td> <td> <a class=param>startActs</a>:[<a class=objref>SemAct</a>+]? <a class=param>start</a>:<a class=nobref>shapeExpr</a>? <a class=param>imports</a>:[<a>IRIREF</a>+]? <a class=param>shapes</a>:[<a class=nobref>ShapeDecl</a>+]? } </td> </tr> </tbody> <tbody class=grammarParagraph> <tr class=borderHack> <td></td> <td colspan=3></td> </tr> <tr> <td colspan=4 style=padding-top:2ex>and a description of the mapping of rules in the production to elements of the <a>ShExJ</a> object:</td> </tr> </tbody> <tbody class=prod> <tr class=params> <td style="border-top:thick solid white;border-bottom:thick solid white"></td> <td colspan=3> <ul> <li><a class=param>startActs</a> comes from <span class=prod><a class=grammarRef href=#prod-startActions>startActions</a></span> production.</li> <li><a class=param>start</a> comes from the <span class=prod><a class=grammarRef href=#prod-start>start</a></span> production.</li> <li><a class=param>shapes</a> come from the <span class=prod><a class=grammarRef href=#prod-shapeExprDecl>shapeExprDecl</a></span> production.</li> </ul> </td> </tr> </tbody> <tbody> <tr style=vertical-align:baseline> <td id=prod-directive>[<span class=prodNo>2</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">directive</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-baseDecl>baseDecl</a></span> | <span class=prod><a class=grammarRef href=#prod-prefixDecl>prefixDecl</a></span> | <span class=prod><a class=grammarRef href=#prod-importDecl>importDecl</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-baseDecl>[<span class=prodNo>3</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">baseDecl</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"BASE" <span class=prod><a class=grammarRef href=#term-IRIREF>IRIREF</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-prefixDecl>[<span class=prodNo>4</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">prefixDecl</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"PREFIX" <span class=prod><a class=grammarRef href=#term-PNAME\_NS>PNAME\_NS</a></span> <span class=prod><a class=grammarRef href=#term-IRIREF>IRIREF</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-importDecl>[<span class=prodNo>4,%¢</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">importDecl</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"IMPORT" <span class=prod><a class=grammarRef href=#term-IRIREF>IRIREF</a></span></code> </td> </tr> <tr class=params> <td></td> <td colspan=3>"<code>IMPORT</code>" is described in <a href=#import>ShEx Import</a>. </td> </tr> <tr style=vertical-align:baseline> <td id=prod-notStartAction>[<span class=prodNo>5</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">notStartAction</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-start>start</a></span> | <span class=prod><a class=grammarRef href=#prod-shapeExprDecl>shapeExprDecl</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-start>[<span class=prodNo>6</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">start</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"start" '=' <span class=prod><a class=grammarRef href=#prod-inlineShapeExpression>inlineShapeExpression</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-startActions>[<span class=prodNo>7</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">startActions</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-codeDecl>codeDecl</a></span>+</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-statement>[<span class=prodNo>8</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">statement</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-directive>directive</a></span> | <span class=prod><a class=grammarRef href=#prod-notStartAction>notStartAction</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-shapeExprDecl>[<span class=prodNo>9</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeExprDecl</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-shapeExprLabel>shapeExprLabel</a></span> (<span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span> | "EXTERNAL")</code> </td> </tr> <tr class=params> <td></td> <td colspan=3>If the "<code>EXTERNAL</code>" keyword is present, <span class=prod><a class=grammarRef href=#prod-shapeExprDecl>shapeExprDecl</a></span> returns a <a class=obj>ShapeDecl</a> object whose <span class=param>shapeExpr</span> is a <a class=obj>ShapeExternal</a>:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>ShapeDecl</a></td> <td>{</td> <td><span class=param>id</span>:<a class=nobref>shapeExprLabel</a> <span class=param>abstract</span>:<a class=nobref>BOOL</a>? <span class=param>shapeExpr</span>: { <span class=param>type</span>: "ShapeExternal" } }</td> </tr> <tr class=params> <td></td> <td colspan=3> <ul> <li><span class=param>abstract</span> is true if "<code>ABSTRACT</code>" was matched.</li> </ul> </td> </tr> <tr class=params> <td></td> <td colspan=3>otherwise <span class=prod><a class=grammarRef href=#prod-shapeExprDecl>shapeExprDecl</a></span> returns a <span class=obj>ShapeDecl</span> object whose <span class=param>shapeExpr</span> is a <span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span>.</td> </tr> <tr class=obj> <td></td> <td><a class=obj>ShapeDecl</a></td> <td>{</td> <td><span class=param>id</span>:<a class=nobref>shapeExprLabel</a> <span class=param>abstract</span>:<a class=nobref>BOOL</a>? <span class=param>shapeExpr</span>: <a class=nobref>shapeExpr</a> }</td> </tr> <tr class=params> <td></td> <td colspan=3> <ul> <li><span class=param>abstract</span> is true if "<code>ABSTRACT</code>" was matched.</li> </ul> </td> </tr> </tbody> <tbody class=grammarParagraph> <tr class=borderHack> <td></td> <td colspan=3></td> </tr> <tr> <td colspan=4> <p> Shape expressions are logical combinations of shape atoms. Inline variants of <a>shape expressions</a> are used in <span class=prod><a class=grammarRef href=#prod-tripleConstraint>tripleConstraint</a></span>s and are not permitted to have annotations or semantic actions. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-shapeExpression>[<span class=prodNo>10</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeExpression</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-shapeOr>shapeOr</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-inlineShapeExpression>[<span class=prodNo>11</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">inlineShapeExpression</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-inlineShapeOr>inlineShapeOr</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-shapeOr>[<span class=prodNo>12</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeOr</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-shapeAnd>shapeAnd</a></span> ("OR" <span class=prod><a class=grammarRef href=#prod-shapeAnd>shapeAnd</a></span>)\*</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-inlineShapeOr>[<span class=prodNo>13</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">inlineShapeOr</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-inlineShapeAnd>inlineShapeAnd</a></span> ("OR" <span class=prod><a class=grammarRef href=#prod-inlineShapeAnd>inlineShapeAnd</a></span>)\*</code> </td> </tr> <tr class=params> <td></td> <td colspan=3> If the right <a class=grammarRef href=#prod-shapeAnd>shapeAnd</a> matches one or more times, the result is a <a class=obj>ShapeOr</a> object with <span class=param>shapeExprs</span> containing the first <a class=grammarRef href=#prod-shapeAnd>shapeAnd</a> followed by the ordered list from the second <a class=grammarRef href=#prod-shapeAnd>shapeAnd</a>: </td> </tr> <tr class=obj> <td></td> <td><a class=obj>ShapeOr</a></td> <td>{</td> <td><span class=param>id</span>:<a class=nobref>shapeExprLabel</a>? <span class=param>shapeExprs</span>:[<a class=nobref>shapeExpr</a>{2,}] }</td> </tr> <tr class=params> <td></td> <td colspan=3> otherwise the result is the left <a class=grammarRef href=#prod-shapeAnd>shapeAnd</a>. </td> </tr> <tr style=vertical-align:baseline> <td id=prod-shapeAnd>[<span class=prodNo>14</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeAnd</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-shapeNot>shapeNot</a></span> ("AND" <span class=prod><a class=grammarRef href=#prod-shapeNot>shapeNot</a></span>)\*</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-inlineShapeAnd>[<span class=prodNo>15</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">inlineShapeAnd</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-inlineShapeNot>inlineShapeNot</a></span> ("AND" <span class=prod><a class=grammarRef href=#prod-inlineShapeNot>inlineShapeNot</a></span>)\*</code> </td> </tr> <tr class=params> <td></td> <td colspan=3> If the right <a class=grammarRef href=#prod-shapeNot>shapeNot</a> matches one or more times, the result is a <a class=obj>ShapeAnd</a> object with <span class=param>shapeExprs</span> containing the first <a class=grammarRef href=#prod-shapeNot>shapeNot</a> followed by the ordered list from the second <a class=grammarRef href=#prod-shapeNot>shapeNot</a>: </td> </tr> <tr class=obj> <td></td> <td><a class=obj>ShapeAnd</a></td> <td>{</td> <td><span class=param>id</span>:<a class=nobref>shapeExprLabel</a>? <span class=param>shapeExprs</span>:[<a class=nobref>shapeExpr</a>{2,}] }</td> </tr> <tr class=params> <td></td> <td colspan=3> otherwise the result is the left <a class=grammarRef href=#prod-shapeNot>shapeNot</a>. </td> </tr> <tr style=vertical-align:baseline> <td id=prod-shapeNot>[<span class=prodNo>16</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeNot</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"NOT"? <span class=prod><a class=grammarRef href=#prod-shapeAtom>shapeAtom</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-inlineShapeNot>[<span class=prodNo>17</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">inlineShapeNot</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"NOT"? <span class=prod><a class=grammarRef href=#prod-inlineShapeAtom>inlineShapeAtom</a></span></code> </td> </tr> <tr class=params> <td></td> <td colspan=3> If the left "<code>NOT</code>" matches, the result is a <a class=obj>ShapeNot</a> object with <span class=param>shapeExpr</span> containing the <a class=grammarRef href=#prod-shapeAtom>shapeAtom</a>:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>ShapeNot</a></td> <td>{</td> <td><span class=param>id</span>:<a class=nobref>shapeExprLabel</a>? <span class=param>shapeExpr</span>:<a class=nobref>shapeExpr</a> }</td> </tr> <tr class=params> <td></td> <td colspan=3> otherwise the result is the <a class=grammarRef href=#prod-shapeAtom>shapeAtom</a>. </td> </tr> </tbody> <tbody class=grammarParagraph> <tr class=borderHack> <td></td> <td colspan=3></td> </tr> <tr> <td colspan=4> <p> Shape atoms are shape references (indicated by "<code>@</code>"), definitions, or nested expressions. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-shapeAtom>[<span class=prodNo>18</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeAtom</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-nonLitNodeConstraint>nonLitNodeConstraint</a></span> <span class=prod><a class=grammarRef href=#prod-shapeOrRef>shapeOrRef</a></span>?<br> | <span class=prod><a class=grammarRef href=#prod-litNodeConstraint>litNodeConstraint</a></span><br> | <span class=prod><a class=grammarRef href=#prod-shapeOrRef>shapeOrRef</a></span> <span class=prod><a class=grammarRef href=#prod-nonLitNodeConstraint>nonLitNodeConstraint</a></span>?<br> | '(' <span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span> ')'<br> | '.'</code></td> </tr> <tr style=vertical-align:baseline> <td id=prod-shapeAtomNoRef>[<span class=prodNo>19</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeAtomNoRef</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-nonLitNodeConstraint>nonLitNodeConstraint</a></span> <span class=prod><a class=grammarRef href=#prod-shapeOrRef>shapeOrRef</a></span>?<br> | <span class=prod><a class=grammarRef href=#prod-litNodeConstraint>litNodeConstraint</a></span><br> | <span class=prod><a class=grammarRef href=#prod-shapeDefinition>shapeDefinition</a></span> <span class=prod><a class=grammarRef href=#prod-nonLitNodeConstraint>nonLitNodeConstraint</a></span>?<br> | '(' <span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span> ')'<br> | '.'</code></td> </tr> <tr style=vertical-align:baseline> <td id=prod-inlineShapeAtom>[<span class=prodNo>20</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">inlineShapeAtom</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-nonLitNodeConstraint>nonLitNodeConstraint</a></span> <span class=prod><a class=grammarRef href=#prod-inlineShapeOrRef>inlineShapeOrRef</a></span>?<br> | <span class=prod><a class=grammarRef href=#prod-litNodeConstraint>litNodeConstraint</a></span><br> | <span class=prod><a class=grammarRef href=#prod-inlineShapeOrRef>inlineShapeOrRef</a></span> <span class=prod><a class=grammarRef href=#prod-nonLitNodeConstraint>nonLitNodeConstraint</a></span>?<br> | '(' <span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span> ')'<br> | '.'</code></td> </tr> <tr class=params> <td></td> <td colspan=3> <ul> <li>If the matching production inludes both a node constraint (<span class=prod><a class=grammarRef href=#prod-litNodeConstraint>litNodeConstraint</a></span> OR <span class=prod><a class=grammarRef href=#prod-nonLitNodeConstraint>nonLitNodeConstraint</a></span>) <span class=param>nc</span> and a <span class=prod><a class=grammarRef href=#prod-shapeOrRef>shapeOrRef</a></span>, the result is a <a class=obj>ShapeAnd</a> object with <span class=param>shapeExprs</span> containing the list of <span class=param>nc</span> and a <span class=prod><a class=grammarRef href=#prod-shapeOrRef>shapeOrRef</a></span>.</li> <li>If the "(" <span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span> ")" production matches, the result is the result of <span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span>.</li> <li>If the <code>"."</code> production matches, the result is an empty shape: <code>{"type": "Shape"}</code>.</li> </ul> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-shapeOrRef>[<span class=prodNo>21</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeOrRef</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-shapeDefinition>shapeDefinition</a></span> | <span class=prod><a class=grammarRef href=#prod-shapeRef>shapeRef</a></span> </code></td> </tr> <tr style=vertical-align:baseline> <td id=prod-inlineShapeOrRef>[<span class=prodNo>22</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">inlineShapeOrRef</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-inlineShapeDefinition>inlineShapeDefinition</a></span> | <span class=prod><a class=grammarRef href=#prod-shapeRef>shapeRef</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-shapeRef>[<span class=prodNo>23</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeRef</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content>( <span class=prod><a class=grammarRef href=#term-ATPNAME\_LN>ATPNAME\_LN</a></span> | <span class=prod><a class=grammarRef href=#term-ATPNAME\_NS>ATPNAME\_NS</a></span> | '@' <span class=prod><a class=grammarRef href=#prod-shapeExprLabel>shapeExprLabel</a></span> ) "EXACTLY"</code> </td> </tr> <tr class=params> <td></td> <td colspan=3> <ul> <li>If the <span class=prod><a class=grammarRef href=#prod-shapeDefinition>shapeDefinition</a></span> production matches, the result is <span class=prod><a class=grammarRef href=#prod-shapeDefinition>shapeDefinition</a></span>.</li> <li>Otherwise, the result is a <a class=obj>ShapeExprRef</a> to <a class=obj>shapeExprLabel</a>.</li> </ul> </td> </tr> <tr class=obj> <td></td> <td><a class=nob>ShapeExprRef</a></td> <td>{</td> <td><span class=param>label</span>: <a class=objref>shapeExprLabel</a> }</td> </tr> <tr class=params> <td></td> <td colspan=3> <ul> <li><span class=param>exact</span> is true if the "<code>EXACTLY</code>" choice was matched.</li> </ul> </tbody> <tbody class=grammarParagraph> <tr class=borderHack> <td></td> <td colspan=3></td> </tr> <tr> <td colspan=4> <p> Node constraints identify a (possibly infinite) set of matching RDF nodes. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-litNodeConstraint>[<span class=prodNo>24</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">litNodeConstraint</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content>"LITERAL" <span class=prod><a class=grammarRef href=#prod-xsFacet>xsFacet</a></span>\*<br> | <span class=prod><a class=grammarRef href=#prod-datatype>datatype</a></span> <span class=prod><a class=grammarRef href=#prod-xsFacet>xsFacet</a></span>\*<br> | <span class=prod><a class=grammarRef href=#prod-valueSet>valueSet</a></span> <span class=prod><a class=grammarRef href=#prod-xsFacet>xsFacet</a></span>\*<br> | <span class=prod><a class=grammarRef href=#prod-numericFacet>numericFacet</a></span>+</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-nonLitNodeConstraint>[<span class=prodNo>25</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">nonLitNodeConstraint</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-nonLiteralKind>nonLiteralKind</a></span> <span class=prod><a class=grammarRef href=#prod-stringFacet>stringFacet</a></span>\*<br> | <span class=prod><a class=grammarRef href=#prod-stringFacet>stringFacet</a></span>+</code> </td> </tr> <tr class=obj> <td></td> <td><a class=obj>NodeConstraint</a></td> <td>{</td> <td><span class=param>id</span>:<a class=nobref>shapeExprLabel</a>? <span class=param>nodeKind</span>:(<span class=literal>"iri"</span> | <span class=literal>"bnode"</span> | <span class=literal>"nonliteral"</span> | <span class=literal>"literal"</span>)? <span class=param>datatype</span>:<a>IRIREF</a>? <a class=nobref>xsFacet</a>\* <span class=param>values</span>:[<a class=nobref>valueSetValue</a>+]? }</td> </tr> <tr style=vertical-align:baseline> <td id=prod-nonLiteralKind>[<span class=prodNo>26</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">nonLiteralKind</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"IRI" | "BNODE" | "NONLITERAL"</code></td> </tr> <tr style=vertical-align:baseline> <td id=prod-xsFacet>[<span class=prodNo>27</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">xsFacet</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-stringFacet>stringFacet</a></span> | <span class=prod><a class=grammarRef href=#prod-numericFacet>numericFacet</a></span></code> </td> </tr> <tr class=obj> <td></td> <td><a class=nob>xsFacet</a></td> <td>=</td> <td><a class=nobref>stringFacet</a> | <a class=nobref>numericFacet</a> ;</td> </tr> <tr style=vertical-align:baseline> <td id=prod-stringFacet>[<span class=prodNo>28</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">stringFacet</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-stringLength>stringLength</a></span> <span class=prod><a class=grammarRef href=#term-INTEGER>INTEGER</a></span><br> | <span class=prod><a class=grammarRef href=#term-REGEXP>REGEXP</a></span></code></td> </tr> <tr style=vertical-align:baseline> <td id=prod-stringLength>[<span class=prodNo>29</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">stringLength</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"LENGTH" | "MINLENGTH" | "MAXLENGTH"</code></td> </tr> <tr class=obj> <td></td> <td><a class=nob>stringFacet</a></td> <td>=</td> <td>(<span class=param>length</span>|<span class=param>minlength</span>|<span class=param>maxlength</span>):<a>INTEGER</a> | <span class=param>pattern</span>:<a>STRING</a> <span class=param>flags</span>:<a>STRING</a>? ;</td> </tr> <tr style=vertical-align:baseline> <td id=prod-numericFacet>[<span class=prodNo>30</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">numericFacet</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-numericRange>numericRange</a></span> <span class=prod><a class=grammarRef href=#prod-numericLiteral>numericLiteral</a></span><br> | <span class=prod><a class=grammarRef href=#prod-numericLength>numericLength</a></span> <span class=prod><a class=grammarRef href=#term-INTEGER>INTEGER</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-numericRange>[<span class=prodNo>31</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">numericRange</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"MININCLUSIVE" | "MINEXCLUSIVE" | "MAXINCLUSIVE" | "MAXEXCLUSIVE"</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-numericLength>[<span class=prodNo>32</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">numericLength</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"TOTALDIGITS" | "FRACTIONDIGITS"</code></td> </tr> <tr class=obj> <td></td> <td><a class=obj>numericFacet</a></td> <td>=</td> <td>(<span class=param>mininclusive</span>|<span class=param>minexclusive</span>|<span class=param>maxinclusive</span>|<span class=param>maxexclusive</span>):<a class=nobref>numericLiteral</a> | (<span class=param>totaldigits</span>|<span class=param>fractiondigits</span>):<a>INTEGER</a> ;</td> </tr> </tbody> <tbody class=grammarParagraph style="border-left:thin solid white;border-right:thin solid white"> <tr class=borderHack> <td></td> <td colspan=3></td> </tr> <tr> <td colspan=4> <p> Shape defintions associate a <a>triple expression</a> with a closed flag and a list of partially constrained (extra) <a data-lt=predicate>predicates</a>. Any <a>predicate</a> appearing in a <a>triple expression</a> is fully constrained unless it appears in the list of extras. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-shapeDefinition>[<span class=prodNo>33</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeDefinition</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>(<span class=prod><a class=grammarRef href=#prod-extraPropertySet>extraPropertySet</a></span> | "CLOSED")\* '{' <span class=prod><a class=grammarRef href=#prod-tripleExpression>tripleExpression</a></span>? '}' <span class=prod><a class=grammarRef href=#prod-annotation>annotation</a></span>\* <span class=prod><a class=grammarRef href=#prod-semanticActions>semanticActions</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-inlineShapeDefinition>[<span class=prodNo>34</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">inlineShapeDefinition</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>(<span class=prod><a class=grammarRef href=#prod-extraPropertySet>extraPropertySet</a></span> | "CLOSED")\* '{' <span class=prod><a class=grammarRef href=#prod-tripleExpression>tripleExpression</a></span>? '}'</code> </td> </tr> <tr class=obj> <td></td> <td><a class=obj>Shape</a></td> <td>{</td> <td> <span class=param>extends</span>:[<a class=objref>shapeExprLabel</a>]? <span class=param>closed</span>:<a>BOOL</a>? <span class=param>extra</span>:[<a>IRIREF</a>+]? <span class=param>expression</span>:<a class=nobref>tripleExpr</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? } </td> </tr> <tr class=params> <td></td> <td colspan=3> <ul> <li><span class=param>closed</span> is true if the "<code>CLOSED</code>" choice was matched one or more times.</li> <li><span class=param>extra</span> is the set of IRIs matching the <a class=grammarRef href=#prod-extraPropertySet>extraPropertySet</a> production.</li> <li><span class=param>expression</span> comes from the <a class=grammarRef href=#prod-tripleExpression>tripleExpression</a> production.</li> <li><span class=param>annotations</span> is the set of <a class=objref>Annotation</a>s matching the <span class=prod><a class=grammarRef href=#prod-annotation>annotation</a></span> production.</li> <li><span class=param>semActs</span> is the set of semantic actions matching the <span class=prod><a class=grammarRef href=#prod-semanticActions>semanticActions</a></span> production.</li> </ul> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-extraPropertySet>[<span class=prodNo>35</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">extraPropertySet</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"EXTRA" <span class=prod><a class=grammarRef href=#prod-predicate>predicate</a></span>+</code> </td> </tr> </tbody> <tbody class=grammarParagraph> <tr> <td colspan=4> <p> Triple expressions are arrangements of triple constraints. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-tripleExpression>[<span class=prodNo>36</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">tripleExpression</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-oneOfTripleExpr>oneOfTripleExpr</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-oneOfTripleExpr>[<span class=prodNo>37</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">oneOfTripleExpr</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-groupTripleExpr>groupTripleExpr</a></span> | <span class=prod><a class=grammarRef href=#prod-multiElementOneOf>multiElementOneOf</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-multiElementOneOf>[<span class=prodNo>38</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">multiElementOneOf</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-groupTripleExpr>groupTripleExpr</a></span> ('|' <span class=prod><a class=grammarRef href=#prod-groupTripleExpr>groupTripleExpr</a></span>)+</code> </td> </tr> <tr class=params> <td></td> <td colspan=3> If the right <a class=grammarRef href=#prod-groupTripleExpr>groupTripleExpr</a> matches one or more times, the result is a <a class=obj>OneOf</a> object with <span class=param>expressions</span> containing the first <a class=grammarRef href=#prod-groupTripleExpr>groupTripleExpr</a> followed by the ordered list from the second <a class=grammarRef href=#prod-groupTripleExpr>groupTripleExpr</a>: </td> </tr> <tr class=obj> <td></td> <td class=obj><a>OneOf</a></td> <td>{</td> <td><span class=param>id</span>:<a class=nobref>tripleExprLabel</a>? <span class=param>expressions</span>:[<a class=nobref>tripleExpr</a>{2,}] <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=params> <td></td> <td colspan=3> otherwise the result is the left <a class=grammarRef href=#prod-groupTripleExpr>groupTripleExpr</a>. </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-groupTripleExpr>[<span class=prodNo>40</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">groupTripleExpr</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-singleElementGroup>singleElementGroup</a></span> | <span class=prod><a class=grammarRef href=#prod-multiElementGroup>multiElementGroup</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-singleElementGroup>[<span class=prodNo>41</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">singleElementGroup</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-unaryTripleExpr>unaryTripleExpr</a></span> ';'?</code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-multiElementGroup>[<span class=prodNo>42</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">multiElementGroup</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-unaryTripleExpr>unaryTripleExpr</a></span> (';' <span class=prod><a class=grammarRef href=#prod-unaryTripleExpr>unaryTripleExpr</a></span>)+ ';'?</code> </td> </tr> <tr class=params> <td></td> <td colspan=3> If the right <a class=grammarRef href=#prod-unaryTripleExpr>unaryTripleExpr</a> matches one or more times, the result is a <a class=obj>EachOf</a> object with <span class=param>expressions</span> containing the first <a class=grammarRef href=#prod-unaryTripleExpr>unaryTripleExpr</a> followed by the ordered list from the second <a class=grammarRef href=#prod-unaryTripleExpr>unaryTripleExpr</a>: </td> </tr> <tr class=obj> <td></td> <td class=obj><a>EachOf</a></td> <td>{</td> <td><span class=param>id</span>:<a class=nobref>tripleExprLabel</a>? <span class=param>expressions</span>:[<a class=nobref>tripleExpr</a>{2,}] <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=params> <td></td> <td colspan=3> otherwise the result is the left <a class=grammarRef href=#prod-unaryTripleExpr>unaryTripleExpr</a>. </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-unaryTripleExpr>[<span class=prodNo>43</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">unaryTripleExpr</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content>('$' <span class=prod><a class=grammarRef href=#prod-tripleExprLabel>tripleExprLabel</a></span>)? (<span class=prod><a class=grammarRef href=#prod-tripleConstraint>tripleConstraint</a></span> | <span class=prod><a class=grammarRef href=#prod-bracketedTripleExpr>bracketedTripleExpr</a></span>)<br> | <span class=prod><a class=grammarRef href=#prod-include>include</a></span></code></td> </tr> <tr style=vertical-align:baseline> <td id=prod-bracketedTripleExpr>[<span class=prodNo>44</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">bracketedTripleExpr</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'(' <span class=prod><a class=grammarRef href=#prod-tripleExpression>tripleExpression</a></span> ')' <span class=prod><a class=grammarRef href=#prod-cardinality>cardinality</a></span>? <span class=prod><a class=grammarRef href=#prod-annotation>annotation</a></span>\* <span class=prod><a class=grammarRef href=#prod-semanticActions>semanticActions</a></span></code> </td> </tr> </tbody> <tbody class=grammarParagraph> <tr> <td colspan=4> <p> Triple constraints are matched against RDF triples. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-tripleConstraint>[<span class=prodNo>45</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">tripleConstraint</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-senseFlags>senseFlags</a></span>? <span class=prod><a class=grammarRef href=#prod-predicate>predicate</a></span> <span class=prod><a class=grammarRef href=#prod-inlineShapeExpression>inlineShapeExpression</a></span> <span class=prod><a class=grammarRef href=#prod-cardinality>cardinality</a></span>? <span class=prod><a class=grammarRef href=#prod-annotation>annotation</a></span>\* <span class=prod><a class=grammarRef href=#prod-semanticActions>semanticActions</a></span></code> </td> </tr> <tr class=obj> <td></td> <td><a class=obj>TripleConstraint</a></td> <td>{</td> <td><span class=param>id</span>:<a>tripleExprLabel</a>? <span class=param>inverse</span>:<a>BOOL</a>? <span class=param>predicate</span>:<a>IRIREF</a> <span class=param>valueExpr</span>:<a class=nobref>shapeExpr</a>? <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=params> <td></td> <td colspan=3> <ul> <li><span class=param>inverse</span> is true if the <span class=prod><a class=grammarRef href=#prod-senseFlags>senseFlags</a></span> matched "<code>^</code>".</li> <li><a class=param>predicate</a> comes from the <a class=grammarRef href=#prod-predicate>predicate</a> production.</li> <li><span class=param>valueExpr</span> comes from the <a class=grammarRef href=#prod-inlineShapeExpression>inlineShapeExpression</a> production. It it is an empty shape <code>{"type": "Shape"}</code>, <span class=param>valueExpr</span> is not assigned.</li> <li><span class=param>min</span> comes from the <a class=grammarRef href=#prod-cardinality>cardinality</a> production.</li> <li><span class=param>max</span> comes from the <a class=grammarRef href=#prod-cardinality>cardinality</a> production.</li> <li><span class=param>annotations</span> is the set of <a class=objref>Annotation</a>s matching the <span class=prod><a class=grammarRef href=#prod-annotation>annotation</a></span> production.</li> <li><span class=param>semActs</span> is the set of semantic actions matching the <span class=prod><a class=grammarRef href=#prod-semanticActions>semanticActions</a></span> production.</li> </ul> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-cardinality>[<span class=prodNo>46</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">cardinality</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'\*' | '+' | '?' | <span class=prod><a class=grammarRef href=#term-REPEAT\_RANGE>REPEAT\_RANGE</a></span></code> </td> </tr> <tr class=params> <td></td> <td colspan=3> In <a>ShExJ</a>, <code>"\*"</code> is represented as <code>-1</code>, standing for the unbounded cardinality.. </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-senseFlags>[<span class=prodNo>47</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">senseFlags</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'^'</code></td> </tr> </tbody> <tbody class=grammarParagraph> <tr> <td colspan=4> <p> Value sets identify ranges of RDF nodes by explicit inclusion or by range (indicated by "<code>~</code>"). Ranges may include exclusions, which may also be ranges but must not in turn contain exclusions. A <a class=param>valueSetValue</a> may be an <a class=param>objectValue</a> or one of <a class=param>IriStem</a>, <a class=param>IriStemRange</a>, <a class=param>LiteralStem</a>, <a class=param>LiteralStemRange</a>, <a class=param>LanguageStem</a>, <a class=param>LanguageStemRange</a>, . </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-valueSet>[<span class=prodNo>48</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">valueSet</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'[' <span class=prod><a class=grammarRef href=#prod-valueSetValue>valueSetValue</a></span>\* ']'</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-valueSetValue>[<span class=prodNo>49</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">valueSetValue</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-iriRange>iriRange</a></span> | <span class=prod><a class=grammarRef href=#prod-literalRange>literalRange</a></span> | <span class=prod><a class=grammarRef href=#prod-languageRange>languageRange</a></span><br> | <span class=prod><a class=grammarRef href=#prod-exclusion>exclusion</a></span>+</code></td> </tr> <tr class=params> <td></td> <td colspan=3>If "<code>.</code>" matches and <span class=prod><a class=grammarRef href=#prod-exclusion>exclusion</a></span> matches one or more times, all matched items must be consistently iri, literal, or language. <code class="production prod">valueSetValue</code> returns either a <a class=obj>IriStemRange</a>, <a class=obj>LiteralStemRange</a>, or <a class=obj>LanguageStemRange</a> object with <span class=param>exclusions</span> equal to the set of results of <span class=prod><a class=grammarRef href=#prod-exclusion>exclusion</a></span>:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>IriStemRange</a></td> <td>{</td> <td><span class=param>stem</span>:(<a>IRIREF</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>IRIREF</a>|<a class=objref>IriStem</a> +] }</td> </tr> <tr class=obj> <td></td> <td><a class=obj>LiteralStemRange</a></td> <td>{</td> <td><span class=param>stem</span>:(<a class=nobref>STRING</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>STRING</a>|<a class=objref>LiteralStem</a> +] }</td> </tr> <tr class=obj> <td></td> <td><a class=obj>LanguageStemRange</a></td> <td>{</td> <td><span class=param>stem</span>:(<a>LANGTAG</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>LANGTAG</a>|<a class=objref>LanguageStem</a> +] }</td> </tr> <tr class=params> <td></td> <td colspan=3>If "<code>~</code>" matches with no <span class=prod><a class=grammarRef href=#prod-exclusion>exclusion</a></span>, <code class="production prod">valueSetValue</code> returns a <a class=obj>Wildcard</a> object:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>Wildcard</a></td> <td>{</td> <td><span class=comment>/\* empty \*/</span> }</td> </tr> <tr style=vertical-align:baseline> <td id=prod-exclusion>[<span class=prodNo>50</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">exclusion</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'.' (<span class=prod><a class=grammarRef href=#prod-iriExclusion>iriExclusion</a></span> | <span class=prod><a class=grammarRef href=#prod-literalExclusion>literalExclusion</a></span> | <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span>) +</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-iriExclusion>[<span class=prodNo>50.1</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">iriExclusion</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'-' <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> '~'?</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-literalExclusion>[<span class=prodNo>50.2</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">literalExclusion</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'-' <span class=prod><a class=grammarRef href=#prod-literal>literal</a></span> '~'?</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-languageExclusion>[<span class=prodNo>50.3</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">languageExclusion</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'-' <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> '~'?</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-iriRange>[<span class=prodNo>51</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">iriRange</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> ('~' <span class=prod><a class=grammarRef href=#prod-iriExclusion>iriExclusion</a></span>\*)?</code> </td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> matches with no "<code>~</code>", iriRange returns <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span>.</td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> and "<code>~</code>" match with no <span class=prod><a class=grammarRef href=#prod-iriExclusion>iriExclusion</a></span>, iriRange returns a <a class=obj>IriStem</a> object:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>IriStem</a></td> <td>{</td> <td><span class=param>stem</span>:<a>IRIREF</a> }</td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> and "<code>~</code>" match and <span class=prod><a class=grammarRef href=#prod-iriExclusion>iriExclusion</a></span> matches one or more times, iriRange returns a <a class=obj>IriStemRange</a> object with <span class=param>exclusions</span> equal to the set of results of <span class=prod><a class=grammarRef href=#prod-iriExclusion>iriExclusion</a></span>:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>IriStemRange</a></td> <td>{</td> <td><span class=param>stem</span>:(<a>IRIREF</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>IRIREF</a>|<a class=objref>IriStem</a> +] }</td> </tr> <tr style=vertical-align:baseline> <td id=prod-iriExclusion>[<span class=prodNo>52</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">iriExclusion</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'-' <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> '~'?</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-literalRange>[<span class=prodNo>53</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">literalRange</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#prod-literal>literal</a></span> ('~' <span class=prod><a class=grammarRef href=#prod-literalExclusion>literalExclusion</a></span>\*)?</code> </td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#prod-literal>literal</a></span> matches with no "<code>~</code>", literalRange returns <span class=prod><a class=grammarRef href=#prod-literal>literal</a></span>.</td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#prod-literal>literal</a></span> and "<code>~</code>" match with no <span class=prod><a class=grammarRef href=#prod-literalExclusion>literalExclusion</a></span>, literalRange returns a <a class=obj>LiteralStem</a> object:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>LiteralStem</a></td> <td>{</td> <td><span class=param>stem</span>:<a>STRING</a> }</td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#prod-literal>literal</a></span> and "<code>~</code>" match and <span class=prod><a class=grammarRef href=#prod-literalExclusion>literalExclusion</a></span> matches one or more times, literalRange returns a <a class=obj>LiteralStemRange</a> object with <span class=param>exclusions</span> equal to the set of results of <span class=prod><a class=grammarRef href=#prod-literalExclusion>literalExclusion</a></span>: </td> </tr> <tr class=obj> <td></td> <td><a class=obj>LiteralStemRange</a></td> <td>{</td> <td><span class=param>stem</span>:(<a class=nobref>STRING</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>STRING</a>|<a class=objref>LiteralStem</a> +] }</td> </tr> <tr style=vertical-align:baseline> <td id=prod-literalExclusion>[<span class=prodNo>54</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">literalExclusion</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'-' <span class=prod><a class=grammarRef href=#prod-literal>literal</a></span> '~'?</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-languageRange>[<span class=prodNo>55</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">languageRange</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> ('~' <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span>\*)?<br> | <span class=prod>'@' '~' <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span>\*</span></code> </td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> matches with no "<code>~</code>" match , languageRange returns a <a class=obj>Language</a> object with <span class=param>languageTag</span> equal to <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span>:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>Language</a></td> <td>{</td> <td><span class=param>languageTag</span>:<a>LANGTAG</a> }</td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> and "<code>~</code>" match with no <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span>, languageRange returns a <a class=obj>LanguageStem</a> object:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>LanguageStem</a></td> <td>{</td> <td><span class=param>stem</span>:<a>LANGTAG</a> }</td> </tr> <tr class=params> <td></td> <td colspan=3>If <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> and "<code>~</code>" match and <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span> matches one or more times, languageRange returns a <a class=obj>LanguageStemRange</a> object with <span class=param>exclusions</span> equal to the set of results of <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span>: </td> </tr> <tr class=obj> <td></td> <td><a class=obj>LanguageStemRange</a></td> <td>{</td> <td><span class=param>stem</span>:(<a>LANGTAG</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>LANGTAG</a>|<a class=objref>LanguageStem</a> +] }</td> </tr> <tr class=params> <td></td> <td colspan=3>If '@' '~' matched with no <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span>, languageRange returns a <a class=obj>LanguageStemRange</a> object with an empty <span class=param>stem</span>:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>LanguageStemRange</a></td> <td>{</td> <td><span class=param>stem</span>: "" }</td> </tr> <tr class=params> <td></td> <td colspan=3>If '@' '~' matched and <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span> matches one or more times, languageRange returns a <a class=obj>LanguageStemRange</a> object with an empty <span class=param>stem</span> ad <span class=param>exclusions</span> equal to the set of results of <span class=prod><a class=grammarRef href=#prod-languageExclusion>languageExclusion</a></span>:</td> </tr> <tr class=obj> <td></td> <td><a class=obj>LanguageStemRange</a></td> <td>{</td> <td><span class=param>stem</span>: "" <span class=param>exclusions</span>:[<a class=nobref>LANGTAG</a>|<a class=objref>LanguageStem</a> +] }</td> </tr> <tr style=vertical-align:baseline> <td id=prod-languageExclusion>[<span class=prodNo>56</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">languageExclusion</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'-' <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> '~'?</code> </td> </tr> </tbody> <tbody class=grammarParagraph> <tr> <td colspan=4> <p> Triple expressions can include the <span class=prod><a class=grammarRef href=#prod-shapeExpression>shapeExpression</a></span> in a <span class=prod><a class=grammarRef href=#prod-shapeExprDecl>shapeExprDecl</a></span>. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-include>[<span class=prodNo>57</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">include</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'&amp;' <span class=prod><a class=grammarRef href=#prod-tripleExprLabel>tripleExprLabel</a></span></code> </td> </tr> <tr class=params> <td></td> <td colspan=3>Per the <a href=#tripleExprRef-requirement>triple expression refrence requirement</a>, <span class=param>tripleExprLabel</span> property MUST appear in the schema's <a class=param>shapes</a> map and the corresponding <a>triple expression</a> MUST be a <span class=jobjref><a>Shape</a></span> with a <span class=param>tripleExpr</span>.</td> </tr> <tr class=obj> <td></td> <td><a class=obj>tripleExprRef</a></td> <td>=</td> <td><a class=nobref>tripleExprLabel</a> ;</td> </tr> </tbody> <tbody class=grammarParagraph> <tr class=borderHack> <td></td> <td colspan=3></td> </tr> <tr> <td colspan=4> <p> Triple expressions can include annotations in the form of a tuple of a <span class=prod><a class=grammarRef href=#prod-predicate>predicate</a></span> and an <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> or <span class=prod><a class=grammarRef href=#prod-literal>literal</a></span>. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-annotation>[<span class=prodNo>58</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">annotation</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"//" <span class=prod><a class=grammarRef href=#prod-predicate>predicate</a></span> (<span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> | <span class=prod><a class=grammarRef href=#prod-literal>literal</a></span>)</code> </td> </tr> <tr class=obj> <td></td> <td><a class=obj>Annotation</a></td> <td>{</td> <td><a class=param>predicate</a>:<a>IRIREF</a> <span class=param>object</span>:<a class=nobref>objectValue</a> }</td> </tr> </tbody> <tbody class=grammarParagraph> <tr class=borderHack> <td></td> <td colspan=3></td> </tr> <tr> <td colspan=4> <p> Triple expressions can include semantic actions consisting of an <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> and an optional code string. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-semanticActions>[<span class=prodNo>59</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">semanticActions</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-codeDecl>codeDecl</a></span>\*</code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-codeDecl>[<span class=prodNo>60</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">codeDecl</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'%' <span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> (<span class=prod><a class=grammarRef href=#term-CODE>CODE</a></span> | '%')</code> </td> </tr> <tr class=obj> <td></td> <td><a class=obj>SemAct</a></td> <td>{</td> <td><span class=param>name</span>:<a>IRIREF</a> <span class=param>code</span>:<a>STRING</a>? }</td> </tr> </tbody> <tbody class=grammarParagraph> <tr class=borderHack> <td></td> <td colspan=3></td> </tr> <tr> <td colspan=4> <p> The remaining productions come from the specifications for SPARQL and Turtle. </p> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-literal>[<span class=prodNo>13t</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">literal</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-rdfLiteral>rdfLiteral</a></span> | <span class=prod><a class=grammarRef href=#prod-numericLiteral>numericLiteral</a></span> | <span class=prod><a class=grammarRef href=#prod-booleanLiteral>booleanLiteral</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-predicate>[<span class=prodNo>61</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">predicate</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> | <span class=prod><a class=grammarRef href=#term-RDF\_TYPE>RDF\_TYPE</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-datatype>[<span class=prodNo>62</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">datatype</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-iri>iri</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-shapeExprLabel>[<span class=prodNo>63</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">shapeExprLabel</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> | <span class=prod><a class=grammarRef href=#prod-blankNode>blankNode</a></span></code> </td> </tr> <tr style=vertical-align:baseline> <td id=prod-tripleExprLabel>[<span class=prodNo>64</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">tripleExprLabel</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-iri>iri</a></span> | <span class=prod><a class=grammarRef href=#prod-blankNode>blankNode</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-numericLiteral>[<span class=prodNo>16t</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">numericLiteral</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-INTEGER>INTEGER</a></span> | <span class=prod><a class=grammarRef href=#term-DECIMAL>DECIMAL</a></span> | <span class=prod><a class=grammarRef href=#term-DOUBLE>DOUBLE</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-rdfLiteral>[<span class=prodNo>65</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">rdfLiteral</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#prod-langString>langString</a></span> | <span class=prod><a class=grammarRef href=#prod-string>string</a></span> ("^^" <span class=prod><a class=grammarRef href=#prod-datatype>datatype</a></span>)?</code> </td> </tr> <tr id=handle-RDFLiteral class=params> <td></td> <td>returns: <a> literal </a></td> <td></td> <td>The literal has a lexical form of the first rule argument, <code>String</code>. If the <code>'^^' iri</code> rule matched, the datatype is <code>iri</code> and the literal has no <a>language tag</a>. If the <code>langString</code> rule matched, the datatype is <code>rdf:langString</code> and the <a>language tag</a> is extracted from <code>langTag</code>. If neither matched, the datatype is <code>xsd:string</code> and the literal has no <a>language tag</a>.</td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-booleanLiteral>[<span class=prodNo>134s</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">booleanLiteral</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"true" | "false"</code></td> </tr> <tr id=handle-BooleanLiteral class=params> <td></td> <td>returns: <a> literal </a></td> <td></td> <td>The literal has a lexical form of the <code>true</code> or <code>false</code>, depending on which matched the input, and a datatype of <code>xsd:boolean</code>.</td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-string>[<span class=prodNo>135s</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">string</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#term-STRING\_LITERAL1>STRING\_LITERAL1</a></span> | <span class=prod><a class=grammarRef href=#term-STRING\_LITERAL\_LONG1>STRING\_LITERAL\_LONG1</a></span><br> | <span class=prod><a class=grammarRef href=#term-STRING\_LITERAL2>STRING\_LITERAL2</a></span> | <span class=prod><a class=grammarRef href=#term-STRING\_LITERAL\_LONG2>STRING\_LITERAL\_LONG2</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-langString>[<span class=prodNo>66</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">langString</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#term-LANG\_STRING\_LITERAL1>LANG\_STRING\_LITERAL1</a></span> | <span class=prod><a class=grammarRef href=#term-LANG\_STRING\_LITERAL\_LONG1>LANG\_STRING\_LITERAL\_LONG1</a></span><br> | <span class=prod><a class=grammarRef href=#term-LANG\_STRING\_LITERAL2>LANG\_STRING\_LITERAL2</a></span> | <span class=prod><a class=grammarRef href=#term-LANG\_STRING\_LITERAL\_LONG2>LANG\_STRING\_LITERAL\_LONG2</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-iri>[<span class=prodNo>136s</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">iri</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-IRIREF>IRIREF</a></span> | <span class=prod><a class=grammarRef href=#prod-prefixedName>prefixedName</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-prefixedName>[<span class=prodNo>137s</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">prefixedName</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-PNAME\_LN>PNAME\_LN</a></span> | <span class=prod><a class=grammarRef href=#term-PNAME\_NS>PNAME\_NS</a></span></code> </td> </tr> </tbody> <tbody class=prod> <tr style=vertical-align:baseline> <td id=prod-blankNode>[<span class=prodNo>138s</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production prod">blankNode</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-BLANK\_NODE\_LABEL>BLANK\_NODE\_LABEL</a></span></code> </td> </tr> </tbody> <tbody class=grammarParagraph> <tr> <td colspan=4> <h3 id=terminals>Terminals</h3> <p> Terminals return: </p> <ul> <li>the RDF abstract types <a class=math>IRI</a>, <a data-cite=rdf11-concepts#dfn-lexical-form>lexical form</a>, <a>literal</a>, <dfn data-cite=rdf11-concepts#dfn-language-tag>language tag</dfn>.</li> <li>a string of unicode codepoints for <a href=#term-CODE>CODE</a>.</li> <li>a <code>repeat range</code> for <a href=#term-REPEAT\_RANGE>REPEAT\_RANGE</a>. A <code>repeat range</code> is a tuple of non-negative integers or a non-negative integer and a token for <code>\*</code>.</li> </ul> </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-CODE>[<span class=prodNo>67</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">CODE</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"{" ([^%\\] | "\\" [%\\] | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>)\* "%" "}"</code> </td> </tr> <tr id=handle-IRIREF-2 class=params> <td></td> <td>returns: a string of unicode codepoints</td> <td></td> <td>The characters between "{" and "%}" are taken, with the <a data-cite=turtle#numeric>numeric escape sequences</a> unescaped, to form the unicode string of the IRI.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-REPEAT\_RANGE>[<span class=prodNo>68</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">REPEAT\_RANGE</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"{" <span class=prod><a class=grammarRef href=#term-INTEGER>INTEGER</a></span> ( "," (<span class=prod><a class=grammarRef href=#term-INTEGER>INTEGER</a></span> | "\*")? )? "}"</code> </td> </tr> <tr id=handle-REPEAT\_RANGE class=params> <td></td> <td>returns: repeat range</td> <td></td> <td>The base-10 numeric values of <span class=prod><a class=grammarRef href=#term-INTEGER>INTEGER</a></span> are taken or a non-negative integer and an <code>\*</code> token if "<code>\*</code>" was matched.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-RDF\_TYPE>[<span class=prodNo>69</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">RDF\_TYPE</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"a"</code></td> </tr> <tr id=handle-RDF\_TYPE class=params> <td></td> <td> returns: <a> IRI </a></td> <td></td> <td>The <code>iri</code> <code>http://www.w3.org/1999/02/22-rdf-syntax-ns#</code> is returned.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-IRIREF>[<span class=prodNo>18t</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">IRIREF</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"&lt;" ([^#0000- &lt;&gt;\"{}|^`\\] | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>)\* "&gt;"</code> </td> </tr> <tr id=handle-IRIREF class=params> <td></td> <td>returns: <a class=math>IRI</a></td> <td></td> <td>The characters between "&lt;" and "&gt;" are taken, with the <a data-cite=turtle#numeric>numeric escape sequences</a> unescaped, to form the unicode string of the IRI. Relative IRI resolution is performed per Turtle <a class=sectionRef data-cite=turtle#sec-iri-references>Section 6.3</a>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PNAME\_NS>[<span class=prodNo>140s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PNAME\_NS</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-PN\_PREFIX>PN\_PREFIX</a></span>? ":"</code> </td> </tr> <tr id=handle-PNAME\_NS class=params> <td></td> <td>returns: <a> PREFIX </a></td> <td></td> <td>When used in a <a href=#prod-prefixDecl>prefixDecl</a> production, the prefix is a potentially empty unicode string matching the first argument of the rule and serves as a key into the <a>prefixes map</a>.</td> </tr> <tr id=handle-PNAME\_NS2 class=params> <td></td> <td> returns: <a> IRI </a></td> <td></td> <td>When used elsewhere, the <code>iri</code> is the value in the <a>prefixes map</a> corresponding to the first argument of the rule.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PNAME\_LN>[<span class=prodNo>141s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PNAME\_LN</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-PNAME\_NS>PNAME\_NS</a></span> <span class=prod><a class=grammarRef href=#term-PN\_LOCAL>PN\_LOCAL</a></span></code> </td> </tr> <tr id=handle-PNAME\_LN class=params> <td></td> <td>returns: <a> IRI </a></td> <td></td> <td>A potentially empty <a>prefix</a> is identified by the first token, <code>PNAME\_NS</code>. The unicode string of the IRI is formed by unescaping the <a href=https://tools.ietf.org/html/rfc3987#page-9>reserved characters</a> [[!rfc7159]] in the second argument, <code>PN\_LOCAL</code>, and concatenating this onto the <code>namespace</code> found in the <a>prefixes map</a>'s entry for <code>PNAME\_NS</code>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-ATPNAME\_NS>[<span class=prodNo>70</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">ATPNAME\_NS</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"@" <span class=prod><a class=grammarRef href=#term-PNAME\_NS>PNAME\_NS</a></span></code> </td> </tr> <tr id=handle-ATPNAME\_NS class=params> <td></td> <td> returns: <a> IRI </a></td> <td></td> <td>The <code>iri</code> is the value in the <a>prefixes map</a> corresponding to the second token of the rule.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-ATPNAME\_LN>[<span class=prodNo>71</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">ATPNAME\_LN</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"@" <span class=prod><a class=grammarRef href=#term-PNAME\_LN>PNAME\_LN</a></span></code> </td> </tr> <tr id=handle-ATPNAME\_LN class=params> <td></td> <td>returns: <a> IRI </a></td> <td></td> <td>A potentially empty <a>prefix</a> is identified by the second token, <code>PNAME\_NS</code>. The unicode string of the IRI is formed by unescaping the <a href=https://tools.ietf.org/html/rfc3987#page-9>reserved characters</a> [[!rfc7159]] in the third token, <code>PN\_LOCAL</code>, and concatenating this onto the <code>namespace</code> found in the <a>prefixes map</a>'s entry for <code>PNAME\_NS</code>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-REGEXP>[<span class=prodNo>72</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">REGEXP</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'/' ([^/\\\n\r]<br> &nbsp;&nbsp;&nbsp;&nbsp; | '\\' [nrt\\|.?\*+(){}$-\[\]^/]<br> &nbsp;&nbsp;&nbsp;&nbsp; | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span><br> &nbsp;&nbsp;&nbsp;&nbsp;)+ '/' [smix]\*</code></td> </tr> <tr class=obj> <td></td> <td></td> <td>{</td> <td><span class=param>pattern</span>:<a>STRING</a> <span class=param>flags</span>:<a>STRING</a>? }</td> </tr> <tr id=handle-REGEXP class=params> <td></td> <td>returns: <a data-cite=ECMASCRIPT-6.0#sec-terms-and-definitions-object>JSON object</a> </td> <td></td> <td><span class=param>pattern</span> is a unicode string formed from the characters between the outermost '/'s by unescaping matches of '\\' '/' in the terminal pattern as well as the <a data-cite=turtle#numeric>numeric escape sequences</a> matched by <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>. The remaining escape sequences are included verbatim in <span class=param>pattern</span>, e.g. <div class="inline shexc"> <pre>^\/\t\\\U0001D4B8$</pre> </div> would become <div class="inline json"> <pre>^/\t\\\U0001D4B8$</pre> </div>.<br> <span class=param>flags</span> is a sequence of the characters [smix] if any were matched. Otherwise no <span class=param>flags</span> attribute is returned. </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-BLANK\_NODE\_LABEL>[<span class=prodNo>142s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">BLANK\_NODE\_LABEL</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"\_:" (<span class=prod><a class=grammarRef href=#term-PN\_CHARS\_U>PN\_CHARS\_U</a></span> | [0-9]) ((<span class=prod><a class=grammarRef href=#term-PN\_CHARS>PN\_CHARS</a></span> | ".")\* <span class=prod><a class=grammarRef href=#term-PN\_CHARS>PN\_CHARS</a></span>)?</code> </td> </tr> <tr id=handle-BLANK\_NODE\_LABEL class=params> <td></td> <td>returns: <a>blank node</a></td> <td></td> <td>The characters following the "<code>\_:</code>" form a <a>blank node identifier</a>. This corresponds to any <a>blank node</a> in the input dataset that had the same label.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-LANGTAG>[<span class=prodNo>145s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">LANGTAG</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"@" ([a-zA-Z])+ ("-" ([a-zA-Z0-9])+)\*</code></td> </tr> <tr id=handle-LANGTAG class=params> <td></td> <td>returns: <a>language tag</a></td> <td></td> <td>The characters following the <code>@</code> form the unicode string of the <a>language tag</a>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-INTEGER>[<span class=prodNo>19t</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">INTEGER</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>[+-]? [0-9]+</code></td> </tr> <tr id=handle-INTEGER class=params> <td></td> <td>returns: <a> literal </a></td> <td></td> <td>The literal has a lexical form of the input string, and a datatype of <code>xsd:integer</code>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-DECIMAL>[<span class=prodNo>20t</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">DECIMAL</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>[+-]? [0-9]\* "." [0-9]+</code></td> </tr> <tr id=handle-DECIMAL class=params> <td></td> <td>returns: <a> literal </a></td> <td></td> <td>The literal has a lexical form of the input string, and a datatype of <code>xsd:double</code>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-DOUBLE>[<span class=prodNo>21t</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">DOUBLE</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>[+-]? ([0-9]+ "." [0-9]\* <span class=prod><a class=grammarRef href=#term-EXPONENT>EXPONENT</a></span> | "."? [0-9]+ <span class=prod><a class=grammarRef href=#term-EXPONENT>EXPONENT</a></span>)</code> </td> </tr> <tr id=handle-DOUBLE class=params> <td></td> <td>returns: <a> literal </a></td> <td></td> <td>The literal has a lexical form of the input string, and a datatype of <code>xsd:double</code>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-EXPONENT>[<span class=prodNo>155s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">EXPONENT</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>[eE] [+-]? [0-9]+</code></td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-STRING\_LITERAL1>[<span class=prodNo>156s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">STRING\_LITERAL1</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"'" ([^'\\\n\r] | <span class=prod><a class=grammarRef href=#term-ECHAR>ECHAR</a></span> | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>)\* "'"</code> </td> </tr> <tr id=handle-STRING\_LITERAL1 class=params> <td></td> <td>returns: <a data-cite=rdf11-concepts#dfn-lexical-form> lexical form</a></td> <td></td> <td>The characters between the outermost "'"s are taken, with <a data-cite=turtle#numeric>numeric</a> and <a data-cite=turtle#string>string</a> escape sequences unescaped, to form the unicode string of a lexical form.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-STRING\_LITERAL2>[<span class=prodNo>157s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">STRING\_LITERAL2</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'"' ([^\"\\\n\r] | <span class=prod><a class=grammarRef href=#term-ECHAR>ECHAR</a></span> | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>)\* '"'</code> </td> </tr> <tr id=handle-STRING\_LITERAL2 class=params> <td></td> <td>returns: <a data-cite=rdf11-concepts#dfn-lexical-form> lexical form</a></td> <td></td> <td>The characters between the outermost '"'s are taken, with <a data-cite=turtle#numeric>numeric</a> and <a data-cite=turtle#string>string</a> escape sequences unescaped, to form the unicode string of a lexical form.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-STRING\_LITERAL\_LONG1>[<span class=prodNo>158s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">STRING\_LITERAL\_LONG1</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"'''" ( ("'" | "''")? ([^\\'\\] | <span class=prod><a class=grammarRef href=#term-ECHAR>ECHAR</a></span> | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>) )\* "'''"</code> </td> </tr> <tr id=handle-STRING\_LITERAL\_LONG1 class=params> <td></td> <td>returns: <a data-cite=rdf11-concepts#dfn-lexical-form> lexical form</a></td> <td></td> <td>The characters between the outermost "'''"s are taken, with <a data-cite=turtle#numeric>numeric</a> and <a data-cite=turtle#string>string</a> escape sequences unescaped, to form the unicode string of a lexical form.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-STRING\_LITERAL\_LONG2>[<span class=prodNo>159s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">STRING\_LITERAL\_LONG2</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'"""' ( ('"' | '""')? ([^\"\\] | <span class=prod><a class=grammarRef href=#term-ECHAR>ECHAR</a></span> | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>) )\* '"""'</code> </td> </tr> <tr id=handle-STRING\_LITERAL\_LONG2 class=params> <td></td> <td>returns: <a data-cite=rdf11-concepts#dfn-lexical-form> lexical form</a></td> <td></td> <td>The characters between the outermost '"""'s are taken, with <a data-cite=turtle#numeric>numeric</a> and <a data-cite=turtle#string>string</a> escape sequences unescaped, to form the unicode string of a lexical form.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-LANG\_STRING\_LITERAL1>[<span class=prodNo>73</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">LANG\_STRING\_LITERAL1</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"'" ([^'\\\n\r] | <span class=prod><a class=grammarRef href=#term-ECHAR>ECHAR</a></span> | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>)\* "'" <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span></code> </td> </tr> <tr id=handle-LANG\_STRING\_LITERAL1 class=params> <td></td> <td>returns: <a data-cite=rdf11-concepts#dfn-lexical-form> lexical form</a></td> <td></td> <td>The characters between the outermost "'"s are taken, with <a data-cite=turtle#numeric>numeric</a> and <a data-cite=turtle#string>string</a> escape sequences unescaped, to form the unicode string of a lexical form. The trailing <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> is used to create a <a>language-tagged string</a>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-LANG\_STRING\_LITERAL2>[<span class=prodNo>74</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">LANG\_STRING\_LITERAL2</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'"' ([^\"\\\n\r] | <span class=prod><a class=grammarRef href=#term-ECHAR>ECHAR</a></span> | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>)\* '"' <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span></code> </td> </tr> <tr id=handle-LANG\_STRING\_LITERAL2 class=params> <td></td> <td>returns: <a data-cite=rdf11-concepts#dfn-lexical-form> lexical form</a></td> <td></td> <td>The characters between the outermost '"'s are taken, with <a data-cite=turtle#numeric>numeric</a> and <a data-cite=turtle#string>string</a> escape sequences unescaped, to form the unicode string of a lexical form. The trailing <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> is used to create a <a>language-tagged string</a>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-LANG\_STRING\_LITERAL\_LONG1>[<span class=prodNo>75</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">LANG\_STRING\_LITERAL\_LONG1</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"'''" ( ("'" | "''")? ([^\\'\\] | <span class=prod><a class=grammarRef href=#term-ECHAR>ECHAR</a></span> | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>) )\* "'''" <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span></code> </td> </tr> <tr id=handle-LANG\_STRING\_LITERAL\_LONG1 class=params> <td></td> <td>returns: <a data-cite=rdf11-concepts#dfn-lexical-form> lexical form</a></td> <td></td> <td>The characters between the outermost "'''"s are taken, with <a data-cite=turtle#numeric>numeric</a> and <a data-cite=turtle#string>string</a> escape sequences unescaped, to form the unicode string of a lexical form. The trailing <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> is used to create a <a>language-tagged string</a>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-LANG\_STRING\_LITERAL\_LONG2>[<span class=prodNo>76</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">LANG\_STRING\_LITERAL\_LONG2</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>'"""' ( ('"' | '""')? ([^\"\\] | <span class=prod><a class=grammarRef href=#term-ECHAR>ECHAR</a></span> | <span class=prod><a class=grammarRef href=#term-UCHAR>UCHAR</a></span>) )\* '"""' <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span></code> </td> </tr> <tr id=handle-LANG\_STRING\_LITERAL\_LONG2 class=params> <td></td> <td>returns: <a data-cite=rdf11-concepts#dfn-lexical-form> lexical form</a></td> <td></td> <td>The characters between the outermost '"""'s are taken, with <a data-cite=turtle#numeric>numeric</a> and <a data-cite=turtle#string>string</a> escape sequences unescaped, to form the unicode string of a lexical form. The trailing <span class=prod><a class=grammarRef href=#term-LANGTAG>LANGTAG</a></span> is used to create a <a>language-tagged string</a>.</td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-UCHAR>[<span class=prodNo>26t</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">UCHAR</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content>"\\u" <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span><br> | "\\U" <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span></code> </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-ECHAR>[<span class=prodNo>160s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">ECHAR</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"\\" [tbnrf\\\"\\']</code></td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PN\_CHARS\_BASE>[<span class=prodNo>164s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PN\_CHARS\_BASE</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content>[A-Z] | [a-z]<br> | [#00C0-#00D6] | [#00D8-#00F6] | [#00F8-#02FF]<br> | [#0370-#037D] | [#037F-#1FFF]<br> | [#200C-#200D] | [#2070-#218F] | [#2C00-#2FEF]<br> | [#3001-#D7FF] | [#F900-#FDCF] | [#FDF0-#FFFD]<br> | [#10000-#EFFFF]</code></td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PN\_CHARS\_U>[<span class=prodNo>165s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PN\_CHARS\_U</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-PN\_CHARS\_BASE>PN\_CHARS\_BASE</a></span> | "\_"</code> </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PN\_CHARS>[<span class=prodNo>167s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PN\_CHARS</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content><span class=prod><a class=grammarRef href=#term-PN\_CHARS\_U>PN\_CHARS\_U</a></span> | "-" | [0-9]<br> | [#00B7] | [#0300-#036F] | [#203F-#2040]</code></td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PN\_PREFIX>[<span class=prodNo>168s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PN\_PREFIX</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-PN\_CHARS\_BASE>PN\_CHARS\_BASE</a></span> ( (<span class=prod><a class=grammarRef href=#term-PN\_CHARS>PN\_CHARS</a></span> | ".")\* <span class=prod><a class=grammarRef href=#term-PN\_CHARS>PN\_CHARS</a></span> )?</code> </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PN\_LOCAL>[<span class=prodNo>77</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PN\_LOCAL</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>(<span class=prod><a class=grammarRef href=#term-PN\_CHARS\_U>PN\_CHARS\_U</a></span> | ":" | [0-9] | <span class=prod><a class=grammarRef href=#term-PLX>PLX</a></span>) ((<span class=prod><a class=grammarRef href=#term-PN\_CHARS>PN\_CHARS</a></span> | "." | ":" | <span class=prod><a class=grammarRef href=#term-PLX>PLX</a></span>)\* (<span class=prod><a class=grammarRef href=#term-PN\_CHARS>PN\_CHARS</a></span> | ":" | <span class=prod><a class=grammarRef href=#term-PLX>PLX</a></span>))? </code> </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PLX>[<span class=prodNo>170s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PLX</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content><span class=prod><a class=grammarRef href=#term-PERCENT>PERCENT</a></span> | <span class=prod><a class=grammarRef href=#term-PN\_LOCAL\_ESC>PN\_LOCAL\_ESC</a></span></code> </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PERCENT>[<span class=prodNo>171s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PERCENT</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"%" <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span> <span class=prod><a class=grammarRef href=#term-HEX>HEX</a></span></code> </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-HEX>[<span class=prodNo>172s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">HEX</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>[0-9] | [A-F] | [a-f]</code></td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PN\_LOCAL\_ESC>[<span class=prodNo>173s</span>]&nbsp;&nbsp;&nbsp;</td> <td>&lt;<code class="production term">PN\_LOCAL\_ESC</code>&gt;</td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td><code class=content>"\\" ( "\_" | "~" | "." | "-" | "!" | "$" | "&amp;" | "'" | "(" | ")" | "\*" | "+" | "," | ";" | "=" | "/" | "?" | "#" | "@" | "%" )</code> </td> </tr> </tbody> <tbody class=term> <tr style=vertical-align:baseline> <td id=term-PASSED\_TOKENS>[<span class=prodNo>98</span>]&nbsp;&nbsp;&nbsp;</td> <td><code class="production directive">PASSED TOKENS</code></td> <td>&nbsp;&nbsp;&nbsp;::=&nbsp;&nbsp;&nbsp;</td> <td>&nbsp;&nbsp;&nbsp;<code class=content>[ \t\r\n]+<br> | "#" [^\r\n]\*<br> | "/\*" ([^\*] | '\*' ([^/] | '\\/'))\* "\*/"</code></td> </tr> </tbody> </table> </div> </div> </section> <section id=shexj class=appendix> <h2>ShEx JSON Syntax (ShExJ)</h2> <p> This section aggregates the <a>JSON grammar</a> rules defined above and includes terminals referenced above. </p> <p> A ShExJ document is a JSON-LD [[!JSON-LD]] document which uses a proscribed structure to define a <a>schema</a> containing <a>shape expressions</a> and <a>triple expressions</a>. A ShExJ document MAY include an <code>@context</code> property referencing <code>http://www.w3.org/ns/shex.jsonld</code>. In the absense of a top-level <code>@context</code>, ShEx Processors MUST act as if a <code>@context</code> property is present with the value <code>http://www.w3.org/ns/shex.jsonld</code>. </p> <p> A ShExJ document can also be thought of as the serialization of an <a>RDF Graph</a> using the Shape Expression Vocabulary [[shex-vocab]] which conforms to the shape defined in <a href=#shexr class=secionRef></a>. Processors MAY interpret a ShExJ document as an RDF Graph. Processors may also transform arbitrary RDF Graphs conforming to <a href=#shexr class=secionRef></a> into ShExJ using a mechanism not described within this specification. </p> <p>In ShExJ, the unbounded cardinality constraint is <code>-1</code>, rather than <code>"\*"</code>.</p> <p> This is the complete grammar for <dfn>ShExJ</dfn>. </p> <div class=shexjTable> <table class=shexj> <tbody><tr class=obj> <th class=obj><a>Schema</a></th> <td>{</td> <td><dfn class=param data-lt=context></dfn>"@context":"http://www.w3.org/ns/shex.jsonld"?<br><a class=param>imports</a>:[<a>IRIREF</a>+]? <span class=param>startActs</span>:[<a class=objref>SemAct</a>+]? <a class=param>start</a>:<a class=nobref>shapeExpr</a>? <a class=param>shapes</a>:[<a class=nobref>ShapeDecl</a>+]? }</td> </tr> <tr class=obj> <th class=obj><a>ShapeDecl</a></th> <td>{</td> <td> <span class=param>id</span>:<a class=nobref>shapeExprLabel</a> <span class=param>abstract</span>:<a class=nobref>BOOL</a>? <span class=param>shapeExpr</span>:<a class=nobref>shapeExpr</a> | <a class=objref>ShapeExternal</a> }</td> </tr> <tr class=nob> <th class=obj><a>shapeExpr</a></th> <td>=</td> <td><a class=objref>ShapeOr</a> | <a class=objref>ShapeAnd</a> | <a class=objref>ShapeNot</a> | <a class=objref>NodeConstraint</a> | <a class=objref>Shape</a> | <a class=objref>ShapeExprRef</a> ;</td> </tr> <tr class=obj> <th class=obj><a>ShapeOr</a></th> <td>{</td> <td><span class=param>shapeExprs</span>:[<a class=nobref>shapeExpr</a>{2,}] }</td> </tr> <tr class=obj> <th class=obj><a>ShapeAnd</a></th> <td>{</td> <td><span class=param>shapeExprs</span>:[<a class=nobref>shapeExpr</a>{2,}] }</td> </tr> <tr class=obj> <th class=obj><a>ShapeNot</a></th> <td>{</td> <td><span class=param>shapeExpr</span>:<a class=nobref>shapeExpr</a> }</td> </tr> <tr class=obj> <th class=obj><a>ShapeExternal</a></th> <td>{</td> <td>}</td> </tr> <tr class=nob> <th class=nob><a>ShapeExprRef</a></th> <td>{</td> <td><span class=param>label</span>:<span class=nobref>shapeExprLabel <span class=param>exact</span>:<a>BOOL</a>? }</span></td> </tr> <tr class=nob> <th class=nob><a>shapeExprLabel</a></th> <td>=</td> <td><a>IRIREF</a> | <a>BNODE</a> ;</td> </tr> <tr class=obj> <th class=obj><a>NodeConstraint</a></th> <td>{</td> <td><span class=param>id</span>:<a class=nobref>shapeExprLabel</a>? <span class=param>nodeKind</span>:(<span class=literal>"iri"</span> | <span class=literal>"bnode"</span> | <span class=literal>"nonliteral"</span> | <span class=literal>"literal"</span>)? <span class=param>datatype</span>:<a>IRIREF</a>? <a class=nobref>xsFacet</a>\* <span class=param>values</span>:[<a class=nobref>valueSetValue</a>+]? }</td> </tr> <tr class=nob> <th class=nob><a>xsFacet</a></th> <td>=</td> <td><a class=nobref>stringFacet</a> | <a class=nobref>numericFacet</a> ;</td> </tr> <tr class=nob> <th class=nob><a>stringFacet</a></th> <td>=</td> <td>(<span class=param>length</span>|<span class=param>minlength</span>|<span class=param>maxlength</span>):<a>INTEGER</a> | <span class=param>pattern</span>:<a>STRING</a> <span class=param>flags</span>:<a>STRING</a>? ; </td> </tr> <tr class=nob> <th class=nob><a>numericFacet</a></th> <td>=</td> <td>(<span class=param>mininclusive</span>|<span class=param>minexclusive</span>|<span class=param>maxinclusive</span>|<span class=param>maxexclusive</span>):<a class=nobref>numericLiteral</a> </td> </tr> <tr> <th></th> <td>|</td> <td>(<span class=param>totaldigits</span>|<span class=param>fractiondigits</span>):<a>INTEGER</a> ;</td> </tr> <tr class=nob> <th class=nob><a>numericLiteral</a></th> <td>=</td> <td><a>INTEGER</a> | <a>DECIMAL</a> | <a>DOUBLE</a> ;</td> </tr> <tr class=nob> <th class=nob><a>valueSetValue</a></th> <td>=</td> <td><a class=nobref>objectValue</a> | <a class=objref>IriStem</a> | <a class=objref>IriStemRange</a> | <a class=objref>LiteralStem</a> | <a class=objref>LiteralStemRange</a> | <a class=objref>Language</a> | <a class=objref>LanguageStem</a> | <a class=objref>LanguageStemRange</a> ;</td> </tr> <tr class=nob> <th class=nob><a>objectValue</a></th> <td>=</td> <td><a>IRIREF</a> | <a>ObjectLiteral</a> ;</td> </tr> <tr class=nob> <th class=nob><a>ObjectLiteral</a></th> <td>{</td> <td><span class=param>value</span>:<a>STRING</a> <span class=param>language</span>:<a>STRING</a>? <span class=param>type</span>:<a>STRING</a>? }</td> </tr> <tr class=obj> <th class=obj><a>IriStem</a></th> <td>{</td> <td><span class=param>stem</span>:<a>IRIREF</a> }</td> </tr> <tr class=obj> <th class=obj><a>IriStemRange</a></th> <td>{</td> <td><span class=param>stem</span>:(<a>IRIREF</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>IRIREF</a>|<a class=objref>IriStem</a>+]? }</td> </tr> <tr class=obj> <th class=obj><a>LiteralStem</a></th> <td>{</td> <td><span class=param>stem</span>:<a>STRING</a> }</td> </tr> <tr class=obj> <th class=obj><a>LiteralStemRange</a></th> <td>{</td> <td><span class=param>stem</span>:(<a class=nobref>STRING</a> | <a class=objref>Wildcard</a>) <span class=param>exclusions</span>:[<a class=nobref>STRING</a>|<a class=objref>LiteralStem</a>+]? }</td> </tr> <tr class=obj> <th class=obj><a>Language</a></th> <td>{</td> <td><span class=param>languageTag</span>:<a>LANGTAG</a> }</td> </tr> <tr class=obj> <th class=obj><a>LanguageStem</a></th> <td>{</td> <td><span class=param>stem</span>:(<a>LANGTAG</a> | <a class=@@ style=background:#FBE9E9>EMPTY</a>) }</td> </tr> <tr class=obj> <th class=obj><a>LanguageStemRange</a></th> <td>{</td> <td><span class=param>stem</span>:(<a>LANGTAG</a> | <a class=@@ style=background:#FBE9E9>EMPTY</a>) <span class=param>exclusions</span>:[<a class=nobref>LANGTAG</a>|<a class=objref>LanguageStem</a>+]? }</td> </tr> <tr class=obj> <th class=obj><a>Wildcard</a></th> <td>{</td> <td><span class=comment>/\* empty \*/</span> }</td> </tr> <tr class=obj> <th class=obj><a>Shape</a></th> <td>{</td> <td><span class=param>extends</span>:[<a class=objref>shapeExprRef</a>]? <span class=param>closed</span>:<a>BOOL</a>? <span class=param>extra</span>:[<a>IRIREF</a>+]? <span class=param>expression</span>:<a class=nobref>tripleExpr</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=nob> <th class=nob><a>tripleExpr</a></th> <td>=</td> <td><a class=objref>EachOf</a> | <a class=objref>OneOf</a> | <a class=objref>TripleConstraint</a> | <a class=objref>tripleExprRef</a> ;</td> </tr> <tr class=obj> <th class=obj><a>EachOf</a></th> <td>{</td> <td><span class=param>id</span>:<a class=nobref>tripleExprLabel</a>? <span class=param>expressions</span>:[<a class=nobref>tripleExpr</a>{2,}] <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=obj> <th class=obj><a>OneOf</a></th> <td>{</td> <td><span class=param>id</span>:<a class=nobref>tripleExprLabel</a>? <span class=param>expressions</span>:[<a class=nobref>tripleExpr</a>{2,}] <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=obj> <th class=obj><a>TripleConstraint</a></th> <td>{</td> <td><span class=param>id</span>:<a class=nobref>tripleExprLabel</a>? <span class=param>inverse</span>:<a>BOOL</a>? <a class=param href=https://www.w3.org/TR/rdf11-concepts/#dfn-predicate>predicate</a>:<a>IRIREF</a> <span class=param>valueExpr</span>:<a class=nobref>shapeExpr</a>? <span class=param>min</span>:<a>INTEGER</a>? <span class=param>max</span>:<a>INTEGER</a>? <span class=param>semActs</span>:[<a class=objref>SemAct</a>+]? <span class=param>annotations</span>:[<a class=objref>Annotation</a>+]? }</td> </tr> <tr class=nob> <th class=nob><a>tripleExprRef</a></th> <td>=</td> <td><a class=objref>tripleExprLabel</a> ;</td> </tr> <tr class=nob> <th class=nob><a>tripleExprLabel</a></th> <td>=</td> <td><a>IRIREF</a> | <a>BNODE</a> ;</td> </tr> <tr class=obj> <th class=obj><a>SemAct</a></th> <td>{</td> <td><span class=param>name</span>:<a>IRIREF</a> <span class=param>code</span>:<a>STRING</a>? } </td> </tr> <tr class=obj> <th class=obj><a>Annotation</a></th> <td>{</td> <td><span class=param><a href=https://www.w3.org/TR/rdf11-concepts/#dfn-predicate>predicate</a></span>:<a href=https://www.w3.org/TR/rdf11-concepts/#dfn-iri>IRIREF</a> <span class=param>object</span>:<a class=nobref>objectValue</a> }</td> </tr> <tr style="background-color:white;border-left:thin solid white;border-right:thin solid white"> <th colspan=1 style=text-align:left><span style=opacity:0># </span>Terminals</th> <td></td> <td>These follow the <a href=http://www.w3.org/TR/2008/PER-xml-20080205/#sec-common-syn>rules for terminals in the XML 1.0 5th Edition</a></td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment><a href=http://www.w3.org/TR/turtle/#grammar-production-IRIREF>Turtle IRIREF</a> without enclosing "&lt;&gt;"s</span></td> </tr> <tr class=trm> <th class=trm><dfn>IRIREF</dfn></th> <td>:</td> <td>(<a>PN\_CHARS</a> | '.' | ':' | '/' | '\\' | '#' | '@' | '%' | '&amp;' | <a>UCHAR</a>)\* ;</td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment><a href=http://www.w3.org/TR/turtle/#grammar-production-BLANK\_NODE\_LABEL>Turtle BLANK\_NODE\_LABEL</a></span></td> </tr> <tr class=trm> <th class=trm><dfn>BNODE</dfn></th> <td>:</td> <td>'\_:' (<a>PN\_CHARS\_U</a> | [0-9]) ((<a>PN\_CHARS</a> | '.')\* <a>PN\_CHARS</a>)? ;</td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment><a href=https://tools.ietf.org/html/rfc7159#section-3>JSON boolean values</a></span></td> </tr> <tr class=trm> <th class=trm><dfn>BOOL</dfn></th> <td>:</td> <td>"true" | "false" ;</td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment><a href=http://www.w3.org/TR/turtle/#grammar-production-INTEGER>Turtle INTEGER</a></span></td> </tr> <tr class=trm> <th class=trm><dfn>INTEGER</dfn></th> <td>:</td> <td>[+-]? [0-9] + ;</td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment><a href=http://www.w3.org/TR/turtle/#grammar-production-DECIMAL>Turtle DECIMAL</a></span></td> </tr> <tr class=trm> <th class=trm><dfn>DECIMAL</dfn></th> <td>:</td> <td>[+-]? [0-9]\* '.' [0-9] + ;</td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment><a href=http://www.w3.org/TR/turtle/#grammar-production-DOUBLE>Turtle DOUBLE</a></span></td> </tr> <tr class=trm> <th class=trm><dfn>DOUBLE</dfn></th> <td>:</td> <td>[+-]? ([0-9] + '.' [0-9]\* <a>EXPONENT</a> | '.' [0-9]+ <a>EXPONENT</a> | [0-9]+ <a>EXPONENT</a>) ;</td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment><a href=https://tools.ietf.org/html/bcp47#section-2.1>BCP47 Language-Tag</a></span></td> </tr> <tr class=trm> <th class=trm><dfn>LANGTAG</dfn></th> <td>:</td> <td>[a-zA-Z]+ ('-' [a-zA-Z0-9]+)\* ;</td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment>any <a href=https://tools.ietf.org/html/rfc7159#section-7>JSON string</a></span></td> </tr> <tr class=trm> <th class=trm><dfn>STRING</dfn></th> <td>:</td> <td>.\* ;</td> </tr> <tr class=trm> <td></td> <td><span class=comment>#</span></td> <td><span class=comment>empty string</span></td> </tr> <tr class="trm @@" style=background:#FBE9E9> <th class=trm><dfn>EMPTY</dfn></th> <td>:</td> <td>^$ ;</td> </tr> <tr style="background-color:white;border-left:thin solid white;border-right:thin solid white"> <th colspan=1 style=text-align:left><span style=opacity:0># </span>Components</th> <td></td> <td>These terminals are referenced by other terminals but not by external productions.</td> </tr> <tr class=trm> <th class=trm><dfn>PN\_PREFIX</dfn></th> <td>:</td> <td><a>PN\_CHARS\_BASE</a> ((<a>PN\_CHARS</a> | '.')\* <a>PN\_CHARS</a>)? ;</td> </tr> <tr class=trm> <th class=trm><dfn>PN\_CHARS\_BASE</dfn></th> <td>:</td> <td>&nbsp;&nbsp;[A-Z] | [a-z] | [\u00C0-\u00D6] | [\u00D8-\u00F6]<br> | [\u00F8-\u02FF] | [\u0370-\u037D] | [\u037F-\u1FFF]<br> | [\u200C-\u200D] | [\u2070-\u218F] | [\u2C00-\u2FEF]<br> | [\u3001-\uD7FF] | [\uF900-\uFDCF] | [\uFDF0-\uFFFD]<br> | [\u10000-\uEFFFF] ;</td> </tr> <tr class=trm> <th class=trm><dfn>PN\_CHARS</dfn></th> <td>:</td> <td><a>PN\_CHARS\_U</a> | '-' | [0-9] | '\u00B7' | [\u0300-\u036F] | [\u203F-\u2040] ;</td> </tr> <tr class=trm> <th class=trm><dfn>PN\_CHARS\_U</dfn></th> <td>:</td> <td><a>PN\_CHARS\_BASE</a> | '\_' ;</td> </tr> <tr class=trm> <th class=trm><dfn>UCHAR</dfn></th> <td>:</td> <td>&nbsp;&nbsp;'\\u' <a>HEX</a> <a>HEX</a> <a>HEX</a> <a>HEX</a><br> | '\\U' <a>HEX</a> <a>HEX</a> <a>HEX</a> <a>HEX</a> <a>HEX</a> <a>HEX</a> <a>HEX</a> <a>HEX</a> ;</td> </tr> <tr class=trm> <th class=trm><dfn>HEX</dfn></th> <td>:</td> <td>[0-9] | [A-F] | [a-f] ;</td> </tr> <tr class=trm> <th class=trm><dfn>EXPONENT</dfn></th> <td>:</td> <td>[eE] [+-]? [0-9]+ ;</td> </tr> </table> </div> </section> <section id=shexr class=appendix> <h2>RDF Representation of ShEx (ShExR)</h2> <p> A <dfn>ShExR</dfn> graph is any <a>RDF Graph</a> which conforms to the following <a>shapes schema</a> and meets the <a href=#schema-requirements>Schema Requirements</a>. Every <a>ShExR</a> document is <a href=https://www.w3.org/TR/rdf11-concepts/#dfn-graph-isomorphism>graph isomorphic</a>[[!rdf11-concepts]] to the <a href=https://www.w3.org/TR/json-ld11-api/#deserialize-json-ld-to-rdf-algorithm>RDF serialization</a>[[!json-ld]] of some <a>ShExJ</a> document. </p> <pre class="example nohighlight" title="ShExR Shape Expression Schema" data-include=ShExR.shex data-oninclude=highlightShExC></pre> </section> <section class="appendix normative"> <h2>IANA Considerations</h2> <p>This section has been submitted to the Internet Engineering Steering Group (IESG) for review, approval, and registration with IANA.</p> <h3>text/shex</h3> <dl> <dt>Type name:</dt> <dd>text</dd> <dt>Subtype name:</dt> <dd>shex</dd> <dt>Required parameters:</dt> <dd>None</dd> <dt>Optional parameters:</dt> <dd>None</dd> <dt>Encoding considerations:</dt> <dd>8-bit text</dd> <dd>ShEx Compact Syntax (ShExC) is a text language which is encoded in UTF-8.</dd> <dt>Security considerations:</dt> <dd>Given that <a>ShExC</a> allows the substitution of long IRIs with short terms, <a>ShExC</a> documents may expand considerably when processed and, in the worst case, the resulting data might consume all of the recipient's resources. Applications should treat any data with due skepticism. </dd> <dt>Interoperability considerations:</dt> <dd>Not Applicable</dd> <dt>Published specification:</dt> <dd><a href=http://shex.io/shex-semantics/>http://shex.io/shex-semantics/</a></dd> <dt>Applications that use this media type:</dt> <dd>Any programming environment that requires the exchange of directed graphs. Implementations of ShEx have been created for JavaScript, Python, Ruby, and Java. </dd> <dt>Fragment Identifier Considerations:</dt> <dd> The structure of a ShEx schema is defined by its representation in JSON per <a href=#shexj>ShEx JSON Syntax (ShExJ)</a>. The JSON-LD context &lt;<a href=http://www.w3.org/ns/shex.jsonld>http://www.w3.org/ns/shex.jsonld</a>&gt; defines the RDF representation (ShExR) for every ShEx schema. A ShEx fragment identifies an instance of either the <a href=#dfn-shapeexpr>shapeExpr</a> or <a href=#dfn-tripleexpr>tripleExpr</a> ShExJ productions, as well as the RDF resource (see <a data-cite=RDF11-CONCEPTS#section-fragID>RDF 1.1 Concepts and Abstract Syntax</a> ,%º6 [[RDF11-CONCEPTS]]) in the corresponding ShExR.</dd> <dt>Restrictions on Usage:</dt> <dd>None</dd> <dt>Provisional Registrations:</dt> <dd>Not Applicable</dd> <dt>Additional information:</dt> <dd> <dl> <dt>Deprecated alias names for this type:</dt> <dd>None</dd> <dt>Magic number(s):</dt> <dd></dd> <dt>File extension(s):</dt> <dd>.shex</dd> <dt>Macintosh file type code(s):</dt> <dd>TEXT</dd> </dl> </dd> <dt>Intended usage:</dt> <dd>Common</dd> <dt>Other Information &amp; Comments:</dt> <dd>None</dd> <dt>Contact Person:</dt> <dd> <dl> <dt>Contact Name:</dt> <dd>Eric Prud'hommeaux</dd> <dt>Contact Email Address:</dt> <dd>eric@w3.org</dd> <dt>Change controller:</dt> <dd>W3C</dd> </dl> </dd> </dl> </section> <section id=security> <h3>Security Considerations</h3> <p> Revealing the structure of an RDF graph can reveal information about the content of conformant data. For instance, a schema with a predicate to describe cancer stage indicates that conforming graphs describe patients with cancer. </p> <p> The process of testing a graph's conformance to a schema may involve many detailed queries which could draw resources to respond to API calls or SPARQL queries. </p> <p> ShEx has an extension mechanism which can, in principle, evalute arbitrary code, possibly as some trusted agent. Such extensions should not be executed if they don't come from a trusted source. </p> <p> Since <a>ShEx</a> is intended to be a pure data exchange format for validating <a>RDF graphs</a>, the <a>ShExJ</a> serialization SHOULD NOT be passed through a code execution mechanism such as JavaScript's <code>eval()</code> function to be parsed. An (invalid) document may contain code that, when executed, could lead to unexpected side effects compromising the security of a system.</p> <p> See also, <a href=#iana-considerations class=sectionRef></a>. </p> </section>