A Declarative Semantics for SNOMED CT Expression Constraints

May 21, 2015

Contents

1 Data Types

1.1 Atomic Data Types

This section identifies the atomic data types that are assumed for the rest of this specification. These types are defined as lexical components in the specification.

```
sctId = digitNonZero 5*17( digit )

term = 1*nonwsNonPipe *( 1*SP 1*nonwsNonPipe )

numericValue = QM stringValue QM / "#" numericValue ( decimalValue / integerValue)

stringValue = QM 1*(anyNonEscapedChar / escapedChar) QM

integerValue = ( ["-"/"+"] digitNonZero *digit ) / zero

decimalValue = integerValue "." 1*digit

nonNegativeIntegerValue = (digitNonZero *digit ) / zero
```

- SCTID – *A SNOMED CT id is used to represent an attribute id or a concept id. The initial digit may not be zero. The smallest number of digits is six, and the maximum is 18.*
- TERM – *The term must be the term from a SNOMED CT description that is associated with the concept identified by the preceding concept identifier. For example, the term could be the preferred description, or the preferred description associated with a particular translation. The term may include valid UTF-8 characters except for the pipe "|" character. The term begins with the first non-whitespace character following the starting "|" character and ends with the last non-whitespace character preceding the next "|" character.*
- STRINGVALUE – *A string value includes one or more of any printable ASCII characters enclosed in quotation marks. Quotes and backslash characters within the string must be preceded by the escape character ("\").*
- $\mathbb{Z}$ – built-in Z equivalent of INTEGERVALUE . *An integer may be positive, negative or zero. Positive integers optionally start with a plus sign ("+"), followed by a non-zero digit followed by zero to many additional digits. Negative integers begin with a minus sign ("-") followed by a non-zero digit and zero to many additional digits.*
- DECIMALVALUE – *A decimal value starts with an integer. This is followed by a decimal point and one to many digits.*
- $\mathbb{N}$ – built-in Z equivalent of NONNEGATIVEINTEGERVALUE
- GROUPID – a role group identifier. (Not explicated in the specification, but needed)

$$[sctId, term, decimalValue, stringValue, groupId]$$

We specifically define some well known identifiers: the *is_a* attribute, the *zero_group* and *attribute_concept*, and *refset_concept* the parents of all attributes and all refsets respectively.

$is\_a : sctId$

$zero\_group : groupId$

$attribute\_concept : sctId$

$refset\_concept : sctId$

1.2 Composite Data Types

concreteValue is in anticipation of the enhanced specification that includes target strings, integers or decimals.

- **concreteValue** – a string, integer or decimal literal
- **target** – the target of a relationship that is either an *sctId* or a *concreteValue*

$concreteValue ::=$
$\quad cv\_string \langle\!\langle stringValue \rangle\!\rangle \mid cv\_integer \langle\!\langle \mathbb{Z} \rangle\!\rangle \mid cv\_decimal \langle\!\langle decimalValue \rangle\!\rangle$

$target ::= t\_sctid \langle\!\langle sctId \rangle\!\rangle \mid t\_concrete \langle\!\langle concreteValue \rangle\!\rangle$

2 The Substrate

A substrate represents the context of an interpretation.

2.1 Substrate Components

2.1.1 Quad

Relationships in the substrate are represented a 4 element tuples or "quads" which consist of a source, attribute, target and role group identifier. The is_a attribute may only appear in the zero group, and the target of an is_a attribute must be a $sctId$ (not a $concreteValue$)

$$\begin{array}{|l|}
\hline
\textit{Quad} \\
\hline
s : sctId \\
a : sctId \\
t : target \\
g : groupId \\
\hline
a = is\_a \Rightarrow (g = zero\_group \wedge t \in \operatorname{ran} t\_sctid) \\
\hline
\end{array}$$

2.1.2 conceptReference

The root of the expression constraint language is concept references:

- CONCEPTREFERENCE – *A conceptReference is represented by a ConceptId optionally followed by a term enclosed by a pair of "|" characters. Whitespace before or after the ConceptId is ignored as is any whitespace between the initial "|" characters and the first non-whitespace character in the term or between the last non-whitespace character and before second "|" character.* TERM *is ignored for the purposes of interpretation.*

- CONCEPTID – *The ConceptId must be a valid SNOMED CT identifier for a concept. The initial digit may not be zero. The smallest number of digits is six, and the maximum is 18.*

- ATTRIBUTENAME – *The attribute name is the name of an attribute (or relationship type) to which a value is applied to refine the meaning of a containing expression. The attribute name is represented in the same way as other concept references. If the attribute name is not known then a wild card may be used to represent any attribute concept in the given substrate.*

conceptReference = conceptId [ws "|" ws term ws "|"]

conceptId = sctId

attributeName = conceptReference / wildCard

$$conceptReference == conceptId \times term[0..1]$$
$$conceptId == sctId$$
$$attributeName ::= ancr\langle\!\langle conceptReference \rangle\!\rangle \mid anwc$$

2.2 Substrate

2.2.1 Substrate Definition

A substrate consists of:

- **concepts** The set of *sctId*s (concepts) that are considered valid in the context of the substrate.
- **relationships** A set of relationship quads (source, attribute, target, group)
- **parentsOf** A function from an sctId to its asserted and inferred parents
- **equivalent_concepts** A function from an sctId to the set of sctId's that have been determined to be equivalent to it.
- **refsets** The reference sets within the context of the substrate whose members are members are concept identifiers (i.e. are in *concepts*). While not formally spelled out in this specification, it is assumed that the typical reference set function would be returning a subset of the *refsetId / referencedComponentId* tuples represented in one or more RF2 Refset Distribution tables.

The following functions can be computed from the basic set above

- **childrenOf** The inverse of the parentsOf function
- **descendants** The transitive closure of the childrenOf function
- **ancestors** The transitive closure of the parentsOf function

A substrate also implements three functions:

- **i_conceptReference** The interpretation of a concept reference. This function can return a (possibly empty) set of sctId's or an error.
- **i_attributeName** The interpretation of an attribute name. This function can return a (possibly empty) set of sctId's or an error.
- **i_refsetId** The interpretation of a refset identifier. This function can return a (possibly empty) set of sctId's or an error.

The formal definition of substrate follows. The expressions below assert that:

1. All *relationship* sources, attributes and sctId targets are in *concepts*.
2. There is a *parentsOf* entry for every substrate *concept*.
3. Every sctId that can be returned by the *parentsOf* function is a *concept*.
4. Every *is_a* relationship entry is represented in the *parentsOf* function. Note that there can be additional entries represented in the *parentsOf* function that aren't in the relationships table.
5. There is an *equivalent_concepts* assertion for every substrate *concept*.
6. The *equivalent_concepts* function is reflexive (i.e. every concept is equivalent to itself).
7. All equivalent concepts are in *concepts*.

8. If two concepts (c2 and c2) are equivalent, then they:
   - Have the same parents
   - Appear the subject, attribute and object of the same set of relationships
   - Appear in the domain of the same set of refsets
   - Both appear in the range of any refset that one appears in
9. Every refset is a substrate *concept*
10. Every member of a refset is a substrate *concept*
11. *childrenOf* is the inverse of *parentsOf*, where any concept that doesn't appear a parent has no (the emptyset) children.
12. *descendants* is the transitive closure of the *childrenOf* function
13. *ancestors* is the transitive closure of the *parentsOf* function
14. No concept can be its own ancestor (or, by inference, descendant)
15. The $i\_conceptReference, i\_attributeName$ and $i\_refsetId$ functions are defined for all possible *conceptReferences* and *attributeNames* (because they are complete functions).
16. All $sctId's$ that are produced by the The $i\_conceptReference, i\_attributeName$ and $i\_refsetId$ functions are substrate *concepts*.

```
┌─ Substrate ────────────────────────────────────────────────
│ concepts : ℙ sctId
│ relationships : ℙ Quad
│ parentsOf : sctId ⇸ ℙ sctId
│ equivalent_concepts : sctId ⇸ ℙ sctId
│ refsets : sctId ⇸ ℙ sctId
│
│ childrenOf : sctId ⇸ ℙ sctId
│ descendants : sctId ⇸ ℙ sctId
│ ancestors : sctId ⇸ ℙ sctId
│
│ groups : ℙ groupId
│ subjects : ℙ sctId
│ targets : ℙ target
│
│ i_conceptReference : conceptReference → Sctids_or_Error
│ i_attributeName : attributeName → Sctids_or_Error
│ i_refsetId : sctId → Sctids_or_Error
├────────────────────────────────────────────────────────────
```

$\forall\, rel : relationships \bullet rel.s \in concepts \land rel.a \in concepts \land$
$\qquad (rel.t \in \operatorname{ran} t\_sctid \Rightarrow t\_sctid^\sim rel.t \in concepts)$

$\operatorname{dom} parentsOf = concepts$
$\bigcup(\operatorname{ran} parentsOf\,) \subseteq concepts$

$\forall\, r : relationships \bullet r.a = is\_a \Rightarrow (t\_sctid^\sim r.t) \in parentsOf\ r.s$

$\operatorname{dom} equivalent\_concepts = concepts$
$\bigcup(\operatorname{ran} equivalent\_concepts) \subseteq concepts$
$\forall\, c : concepts \bullet c \in equivalent\_concepts\ c$
$\forall\, c1, c2 : concepts \mid c2 \in (equivalent\_concepts\ c1) \bullet$
$\qquad parentsOf\ c1 = parentsOf\ c2\ \land$
$\qquad \{r : relationships \mid r.s = c1\} = \{r : relationships \mid r.s = c2\}\ \land$
$\qquad \{r : relationships \mid r.a = c1\} = \{r : relationships \mid r.a = c2\}\ \land$
$\qquad \{r : relationships \mid t\_sctid^\sim r.t = c1\} = \{r : relationships \mid t\_sctid^\sim r.t = c2\}\ \land$
$\qquad c1 \in \operatorname{dom} refsets \Leftrightarrow c2 \in \operatorname{dom} refsets\ \land$
$\qquad c1 \in \operatorname{dom} refsets \Rightarrow refsets\ c1 = refsets\ c2\ \land$
$\qquad (\forall\, rsd : \operatorname{ran} refsets \bullet c1 \in rsd \Leftrightarrow c2 \in rsd)$

$\operatorname{dom} refsets \subseteq concepts$
$\bigcup(\operatorname{ran} refsets) \subseteq concepts$

$\operatorname{dom} childrenOf = concepts$
$\forall\, s, t : concepts \bullet t \in parentsOf\ s \Leftrightarrow s \in childrenOf\ t$
$\forall\, c : concepts \mid c \notin \bigcup(\operatorname{ran} childrenOf\,) \bullet childrenOf\ c = \emptyset$
$\forall\, s : concepts \bullet$
$\qquad descendants\ s = childrenOf\ s \cup \bigcup\{t : childrenOf\ s \bullet descendants\ t\}$
$\forall\, t : concepts \bullet$
$\qquad ancestors\ t = parentsOf\ t \cup \bigcup\{s : parentsOf\ t \bullet ancestors\ s\}$
$\forall\, t : concepts \bullet t \notin ancestors\ t$

$groups = \{r : relationships \bullet r.g\}$
$subjects = \{r : relationships \bullet r.s\}$
$targets = \{r : relationships \bullet r.t\}$

$\forall\, cr\_interp : \operatorname{ran} i\_conceptReference \mid cr\_interp \in \operatorname{ran} ok \bullet$
$\qquad result\_sctids\ cr\_interp \subseteq concepts$
$\forall\, att\_interp : \operatorname{ran} i\_attributeName \mid att\_interp \in \operatorname{ran} ok \bullet$
$\qquad result\_sctids\ att\_interp \subseteq concepts$
$\forall\, refset\_interp : \operatorname{ran} i\_refsetId \mid refset\_interp \in \operatorname{ran} ok \bullet$
$\qquad result\_sctids\ refset\_interp \subseteq concepts$

Implementations may choose to implement "strict" substrates, where additional rules apply or "permissive" substrates where rules are relaxed.

2.2.2 Strict Substrate

A **strict_substrate** is a substrate where:
- If a conceptReference is not in the substrate concepts it returns an error, otherwise the set of equivalent concepts
- If an attribute name is a conceptReference and it is not in the substrate concepts or is not a descendant of the attribute_concept it returns an error, otherwise the set of equivalent attributes. If an attribute name is a wild card, it returns all descendants of attribute_concept
- If a concept reference that is a target of a memberOf function is not in the substrate concepts or is not a descendant of the refset_concept it returns an error, otherwise the set of equivalent refset identifiers

---

strict_substrate
Substrate

$\forall\, cr : conceptReference \bullet i\_conceptReference\ cr =$
    if $first\ cr \notin concepts$ **then** $error\ unknownConceptReference$
    else $ok\,(equivalent\_concepts\,(first\ cr))$

$\forall\, an : attributeName \bullet i\_attributeName\ an =$
if $ancr^{\sim} an \in conceptReference$ **then**
    (**let** $rslt == i\_conceptReference\,(ancr^{\sim} an) \bullet$
        if $rslt \in ran\ error$ **then** $rslt$
        else if $result\_sctids\ rslt \subseteq (descendants\ attribute\_concept)$
            then $rslt$
        else
            $error\ unknownAttributeId)$
else
    $ok\,(descendants\ attribute\_concept)$

$\forall\, rsid : sctId \bullet i\_refsetId\ rsid =$
    if $rsid \in descendants\ refset\_concept$
        then $ok\,\{rsid\}$
    else $error\ unknownRefsetId$

2.2.3 Permissive Substrate

A **permissive_substrate** never raises an error condition. concept references that are not in the substrate are "discarded" (i.e. map to the empty set). Any valid conceptReference may be used in the position of an attribute or a refset id. The attributeName wild card returns the set of all possible concepts.

$\underline{\quad permissive\_substrate \quad}$

 Substrate

$\forall\, cr : conceptReference \bullet i\_conceptReference\ cr =$
  if $first\ cr \notin concepts$ **then** $ok\ \emptyset$
  else $ok\ (equivalent\_concepts\ (first\ cr))$

$\forall\, an : attributeName \bullet i\_attributeName\ an =$
if $ancr^{\sim}\, an \in conceptReference$ **then**
  $i\_conceptReference\ (ancr^{\sim}\, an)$
else
  $ok\ concepts$

$\forall\, rsid : sctId \bullet i\_refsetId\ rsid =$
  if $rsid \notin concepts$ **then** $ok\ \emptyset$
  else $ok\ (equivalent\_concepts\ (rsid))$

3 Interpretation of Expression Constraints

An EXPRESSIONCONSTRAINT is interpreted in the context of a *Substrate* and returns a set of *sctId*s or an error indicator.

3.1 Interpretation Output

The result of applying a query against a substrate is either a (possibly empty) set of sctId's or an *ERROR*. An *ERROR* occurs when:

- The interpretation of a conceptId is not a substrate *concept*
- The interpretation of a relationship attribute is not a substrate *attributeId*
- The interpretation of a reset is not a substrate *refsetId*

$ERROR ::= unknownConceptReference \mid unknownAttributeId \mid unknownRefsetId$

$Sctids\_or\_Error ::= ok\langle\!\langle \mathbb{P}\, sctId \rangle\!\rangle \mid error\langle\!\langle ERROR \rangle\!\rangle$

Each interpretation that follows begins with a simplified version of the language construct in the specification. It then formally specifies the constructs that are used in the interpretation, followed by the interpretation itself. We start with the definition of *expressionConstraint*, which, once interpreted, returns either a set of sctIds or an error condition.

3.2 expressionConstraint

An expression constraint is either a refined expression constraint or an unrefined expression constraint.

---

expressionConstraint = ws (refinedExpressionConstraint / unrefinedExpressionConstraint)
ws

The interpretation of an EXPRESSIONCONSTRAINT is the interpretation of either the REFINEDEXPRESSIONCONSTRAINT or the UNREFINEDEXPRESSIONCONSTRAINT .

$expressionConstraint ::=$
$\qquad expcons\_refined\langle\!\langle refinedExpressionConstraint \rangle\!\rangle \mid$
$\qquad expcons\_unrefined\langle\!\langle unrefinedExpressionConstraint \rangle\!\rangle$

$i\_expressionConstraint :$
$\qquad Substrate \rightarrow expressionConstraint \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ ec : expressionConstraint \bullet i\_expressionConstraint\ ss\ ec =$
if $ec \in \mathrm{ran}\ expcons\_refined$
then
$\qquad i\_refinedExpressionConstraint\ ss\ (expcons\_refined^\sim ec)$
else
$\qquad i\_unrefinedExpressionConstraint\ ss\ (expcons\_unrefined^\sim ec)$

11

3.2.1 unrefinedExpressionConstraint

An unrefined expression constraint is either compound or simple.

unrefinedExpressionConstraint = compoundExpressionConstraint / simpleExpressionConstraint

The interpretation of an UNREFINEDEXPRESSIONCONSTRAINT is either the interpretation of a COMPOUNDEXPRESSIONCONSTRAINT or a SIMPLEEXPRESSIONCONSTRAINT

$unrefinedExpressionConstraint ::=$
$\qquad unrefined\_compound\langle\!\langle compoundExpressionConstraint\rangle\!\rangle \mid$
$\qquad unrefined\_simple\langle\!\langle simpleExpressionConstraint\rangle\!\rangle$

$i\_unrefinedExpressionConstraint :$
$\qquad Substrate \rightarrow unrefinedExpressionConstraint \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ uec : unrefinedExpressionConstraint \bullet$
$i\_unrefinedExpressionConstraint\ ss\ uec =$
if $uec \in \mathrm{ran}\ unrefined\_compound$
then
$\qquad i\_compoundExpressionConstraint\ ss\ (unrefined\_compound^{\sim} uec)$
else
$\qquad i\_simpleExpressionConstraint\ ss\ (unrefined\_simple^{\sim} uec)$

3.2.2 refinedExpressionConstraint

A refined expression constraint starts with an unrefined expression constraint and adds a refinement. Refined expression constraints may optionally be placed in brackets.

refinedExpressionConstraint = unrefinedExpressionConstraint ws ":" ws refinement / "(" ws refinedExpressionConstraint ws ")"

The interpretation of REFINEDEXPRESSIONCONSTRAINT is the <u>intersection</u> of the interpretation of the UNREFINEDEXPRESSIONCONSTRAINT and the REFINEMENT , both of which return a set of sctId's or an error. The second production defines REFINEDEXPRESSIONCONSTRAINT in terms of itself and has no effect on the results.

Note: The second declaration below (*refinedExpressionConstraint'*) is used to avoid circular definitions.

$$refinedExpressionConstraint ==$$
$$unrefinedExpressionConstraint \times refinement$$
$$[refinedExpressionConstraint']$$

$i\_refinedExpressionConstraint :$
    $Substrate \rightarrow refinedExpressionConstraint \rightarrow Sctids\_or\_Error$

$\forall ss : Substrate;\ rec : refinedExpressionConstraint \bullet$
$(\textbf{let } unref\_interp == (i\_unrefinedExpressionConstraint\ ss\ (first\ rec)) \bullet$
    $i\_refinedExpressionConstraint\ ss\ rec =$
        $intersect\ unref\_interp(i\_refinement\ ss\ (second\ rec)))$

Duplicate signature for recursive definitions

$i\_refinedExpressionConstraint' :$
    $Substrate \rightarrow refinedExpressionConstraint' \rightarrow Sctids\_or\_Error$

3.2.3   simpleExpressionConstraint

A simple expression constraint includes exactly one focus concept, optionally preceded by a constraint operator.

simpleExpressionConstraint = [constraintOperator ws] focusConcept

The interpretation of SIMPLEEXPRESSIONCONSTRAINT is the application of an optional constraint operator to the interpretation of FOCUSCONCEPT , which returns a set of sctId's or an error. The interpretation of an error is the error.

$$simpleExpressionConstraint == constraintOperator[0\,..\,1] \times focusConcept$$

$i\_simpleExpressionConstraint :$
    $Substrate \rightarrow simpleExpressionConstraint \rightarrow Sctids\_or\_Error$

$\forall ss : Substrate;\ sec : simpleExpressionConstraint \bullet$
$i\_simpleExpressionConstraint\ ss\ sec =$
    $i\_constraintOperator\ ss\ (first\ sec)\ (i\_focusConcept\ ss\ (second\ sec))$

3.2.4   compoundExpressionConstraint

A compound expression constraint contains two or more simple expression constraints joined by either a conjunction, disjunction or exclusion. Compound expression constraints may optionally be placed in brackets.

> compoundExpressionConstraint = conjunctionExpressionConstraint / disjunctionExpression-
> Constraint / exclusionExpressionConstraint / "(" ws compoundExpressionConstraint ws ")"

The interpretation of a *compoundExpressionConstraint* is the interpretation the conjunction, disjunction or exclusion of nested compound constraint.

$$compoundExpressionConstraint ::=$$
$$compound\_conj \langle\!\langle conjunctionExpressionConstraint \rangle\!\rangle \mid$$
$$compound\_disj \langle\!\langle disjunctionExpressionConstraint \rangle\!\rangle \mid$$
$$compound\_excl \langle\!\langle exclusionExpressionConstraint \rangle\!\rangle \mid$$
$$compound\_nested \langle\!\langle compoundExpressionConstraint \rangle\!\rangle$$
$$[compoundExpressionConstraint']$$

$i\_compoundExpressionConstraint :$
$\qquad Substrate \rightarrow compoundExpressionConstraint \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ cec : compoundExpressionConstraint \bullet$
$i\_compoundExpressionConstraint\ ss\ cec =$
if $cec \in \mathrm{ran}\ compound\_conj$ **then**
$\qquad i\_conjunctionExpressionConstraint\ ss\ (compound\_conj^\sim cec)$
else if $cec \in \mathrm{ran}\ compound\_disj$ **then**
$\qquad i\_disjunctionExpressionConstraint\ ss\ (compound\_disj^\sim cec)$
else if $cec \in \mathrm{ran}\ compound\_excl$ **then**
$\qquad i\_exclusionExpressionConstraint\ ss\ (compound\_excl^\sim cec)$
else
$\qquad i\_compoundExpressionConstraint\ ss\ (compound\_nested^\sim cec)$

$i\_compoundExpressionConstraint' :$
$\qquad Substrate \rightarrow compoundExpressionConstraint' \rightarrow Sctids\_or\_Error$

3.2.5   conjunctionExpressionConstraint

> *A conjunction expression constraint combines two or more expression con-straints with a conjunction ("and") operator. More than one conjunction may be used without brackets. However any compound expression constraint (using a different binary operator) that appears within a conjunction expression constraint must be enclosed by brackets.*

> conjunctionExpressionConstraint = subExpressionConstraint 1*(ws conjunction ws subEx-pressionConstraint)

CONJUNCTIONEXPRESSIONCONSTRAINT is interpreted as the <u>intersection</u> of the interpretations of the SUBEXPRESSIONCONSTRAINTS .

$$conjunctionExpressionConstraint ==$$
$$subExpressionConstraint \times \text{seq}_1(subExpressionConstraint)$$

Apply the intersection operator to the interpretation of each subExpression-Constraint

$i\_conjunctionExpressionConstraint :$
    $Substrate \rightarrow conjunctionExpressionConstraint \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ cecr : conjunctionExpressionConstraint \bullet$
$i\_conjunctionExpressionConstraint\ ss\ cecr =$
    $applyToSequence\ ss\ i\_subExpressionConstraint\ intersect\ cecr$

3.2.6 disjunctionExpressionConstraint

A disjunction expression constraint combines two or more expression constraints with a disjunction ("or") operator. More than one disjunction may be used without brackets. However any compound expression constraint (using a different binary operator) that appears within a disjunction expression constraint must be enclosed by brackets.

disjunctionExpressionConstraint = subExpressionConstraint 1*(ws disjunction ws subExpressionConstraint)

$$disjunctionExpressionConstraint ==$$
$$subExpressionConstraint \times \text{seq}_1(subExpressionConstraint)$$

DISJUNCTIONEXPRESSIONCONSTRAINT is interpreted as the <u>union</u> of the interpretations of the SUBEXPRESSIONCONSTRAINTS .

$i\_disjunctionExpressionConstraint :$
    $Substrate \rightarrow disjunctionExpressionConstraint \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ decr : disjunctionExpressionConstraint \bullet$
$i\_disjunctionExpressionConstraint\ ss\ decr =$
    $applyToSequence\ ss\ i\_subExpressionConstraint\ union\ decr$

3.2.7 exclusionExpressionConstraint

An exclusion expression constraint combines two expression constrains with an exclusion ("minus") operator. A single exclusion operator may be used without brackets. However when the operands of the exclusion expression constraint are compound, these compound expression constraints must be enclosed by brackets.

exclusionExpressionConstraint = subExpressionConstraint ws exclusion ws subExpression-Constraint

EXCLUSIONESPRESSIONCONSTRAINT is interpreted as the interpretation of the SUBEXPRESSIONCONSTRAINT <u>minus</u> the interpretation of the second.

$$exclusionExpressionConstraint ==$$
$$subExpressionConstraint \times subExpressionConstraint$$

$i\_exclusionExpressionConstraint :$
$\qquad Substrate \rightarrow exclusionExpressionConstraint \rightarrow Sctids\_or\_Error$

$\forall ss : Substrate; \ ecr : exclusionExpressionConstraint \bullet$
$(\textbf{let } first\_sec == (i\_subExpressionConstraint \ ss \ (first \ ecr));$
$\qquad second\_sec == (i\_subExpressionConstraint \ ss \ (second \ ecr)) \bullet$
$i\_exclusionExpressionConstraint \ ss \ ecr = minus \ first\_sec \ second\_sec)$

3.2.8 subExpressionConstraint

A subexpression constraint, which appears within a compound expression constraint, must either be simple, or a bracketed compound or refined expression constraint.

subExpressionConstraint = simpleExpressionConstraint / "(" ws (compoundExpressionCon-straint / refinedExpressionConstraint) ws ")"

The interpretation of SUBEXPRESSIONCONSTRAINT is the interpretation the SIMPLEEXPRESSIONCONSTRAINT COMPOUNDEXPRESSIONCONSTRAINT or REFINEDEXPRESSIONCONSTRAINT

$subExpressionConstraint ::=$
$\qquad subExpr\_simple \langle\!\langle simpleExpressionConstraint \rangle\!\rangle \ |$
$\qquad subExpr\_compound \langle\!\langle compoundExpressionConstraint' \rangle\!\rangle \ |$
$\qquad subExpr\_refined \langle\!\langle refinedExpressionConstraint' \rangle\!\rangle$

$i\_subExpressionConstraint :$
$\qquad Substrate \rightarrow subExpressionConstraint \rightarrow Sctids\_or\_Error$

$\forall ss : Substrate; \ sec : subExpressionConstraint \bullet$
$i\_subExpressionConstraint \ ss \ sec =$
$\textbf{if } sec \in \text{ran } subExpr\_simple \textbf{ then}$
$\qquad i\_simpleExpressionConstraint \ ss \ (subExpr\_simple^\sim sec)$
$\textbf{else if } sec \in \text{ran } subExpr\_compound \textbf{ then}$
$\qquad i\_compoundExpressionConstraint' \ ss \ (subExpr\_compound^\sim sec)$
$\textbf{else}$
$\qquad i\_refinedExpressionConstraint' \ ss \ (subExpr\_refined^\sim sec)$

3.3 refinement

A refinement contains all the grouped and ungrouped attributes that refine the set of clinical meanings satisfied by the expression constraint. Refinements may represent the conjunction or disjunction of two smaller refinements, and may optionally be placed in brackets. Where both conjunction and disjunction are used, brackets are mandatory to disambiguate the intended meaning.

refinement = subRefinement ws [conjunctionRefinementSet / disjunctionRefinementSet]

The interpretation of REFINEMENT is one of:
- The <u>intersection</u> of the interpretation of the SUBREFINEMENT and the CONJUNCTIONREFINEMENTSET
- The <u>union</u> of the interpretation of the SUBREFINEMENT and the DISJUNCTIONREFINEMENTSET
- The interpretation of SUBREFINEMENT if neither conjunction nor disjunction set is present

$refinement == subRefinement \times refinementConjunctionOrDisjunction[0 .. 1]$
$[refinement']$

$refinementConjunctionOrDisjunction ::=$
$\quad refine\_conjset \langle\!\langle conjunctionRefinementSet \rangle\!\rangle \mid$
$\quad refine\_disjset \langle\!\langle disjunctionRefinementSet \rangle\!\rangle$

$i\_refinement : Substrate \rightarrow refinement \rightarrow Sctids\_or\_Error$

$\forall ss : Substrate; \ rfnment : refinement \bullet$
$i\_refinement \ ss \ rfnment =$
$(\textbf{let} \ lhs == i\_subRefinement \ ss \ (first \ rfnment); \ rhs == second \ rfnment \bullet$
$\textbf{if} \ \#rhs = 0 \ \textbf{then}$
$\quad lhs$
$\textbf{else if} \ (head \ rhs) \in \text{ran} \ refine\_conjset \ \textbf{then}$
$\quad intersect \ lhs \ (i\_conjunctionRefinementSet \ ss \ (refine\_conjset^\sim(head \ rhs)))$
$\textbf{else}$
$\quad union \ lhs \ (i\_disjunctionRefinementSet \ ss \ (refine\_disjset^\sim(head \ rhs))))$

$i\_refinement' : Substrate \rightarrow refinement' \rightarrow Sctids\_or\_Error$

3.3.1 conjunctionRefinementSet

A conjunction refinement set consists of one or more conjunction operators, each followed by a subRefinement.

```
conjunctionRefinementSet = 1*(ws conjunction ws subRefinement)
```

$$conjunctionRefinementSet == \text{seq}_1\ subRefinement$$

The interpretation of a CONJUNCTIONREFINEMENTSET is the <u>intersection</u> of the interpretation of the SUBREFINEMENTS .

$i\_conjunctionRefinementSet :$
    $Substrate \rightarrow conjunctionRefinementSet \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ conjset : conjunctionRefinementSet \bullet$
$i\_conjunctionRefinementSet\ ss\ conjset =$
if $tail\ conjset = \langle\rangle$ **then**
    $i\_subRefinement\ ss\ (head\ conjset)$
else
    $intersect\ (i\_subRefinement\ ss\ (head\ conjset))\ (i\_conjunctionRefinementSet\ ss\ (tail\ conjset))$

3.3.2 disjunctionRefinementSet

```
disjunctionRefinementSet = 1*(ws disjunction ws subRefinement)
```

$$disjunctionRefinementSet == \text{seq}_1\ subRefinement$$

The interpretation of a DISJUNCTIONREFINEMENTSET is the <u>union</u> of the interpretation of the SUBREFINEMENTS .

$i\_disjunctionRefinementSet :$
    $Substrate \rightarrow disjunctionRefinementSet \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ disjset : disjunctionRefinementSet \bullet$
$i\_disjunctionRefinementSet\ ss\ disjset =$
if $tail\ disjset = \langle\rangle$ **then**
    $i\_subRefinement\ ss\ (head\ disjset)$
else
    $union\ (i\_subRefinement\ ss\ (head\ disjset))\ (i\_disjunctionRefinementSet\ ss\ (tail\ disjset))$

3.3.3 subRefinement

A subRefinement is either an attribute set, an attribute group or a bracketed refinement.

```
subRefinement = attributeSet / attributeGroup / "(" ws refinement ws ")"
```

The interpretation of a SUBREFINEMENT is the interpretation of the corresponding ATTRIBUTESET, ATTRIBUTEGROUP or REFINEMENT.

$subRefinement ::=$
    $subrefine\_attset \langle\!\langle attributeSet \rangle\!\rangle \mid$
    $subrefine\_attgroup \langle\!\langle attributeGroup \rangle\!\rangle \mid$
    $subrefine\_refinement \langle\!\langle refinement' \rangle\!\rangle$

$i\_subRefinement :$
        $Substrate \rightarrow subRefinement \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ subrefine : subRefinement \bullet$
$i\_subRefinement\ ss\ subrefine =$
if $subrefine \in \mathrm{ran}\ subrefine\_attset$ **then**
    $i\_attributeSet\ ss\ (subrefine\_attset^{\sim}\ subrefine)$
else if $subrefine \in \mathrm{ran}\ subrefine\_attgroup$ **then**
    $i\_attributeGroup\ ss\ (subrefine\_attgroup^{\sim}\ subrefine)$
else
    $i\_refinement'\ ss\ (subrefine\_refinement^{\sim}\ subrefine)$

3.4 attributeSet

An attribute set contains one or more attribute name-value pairs separated by a conjunction or disjunction operator. An attribute set may optionally be placed in brackets.

attributeSet = subAttributeSet ws [conjunctionAttributeSet / disjunctionAttributeSet]

The interpretation of an ATTRIBUTESET is one of:
- The <u>intersection</u> of the interpretation of the SUBATTRIBUTESET and the CONJUNCTIONATTRIBUTESET
- The <u>union</u> of the interpretation of the SUBATTRIBUTESET and the DISJUNCTIONATTRIBUTESET
- The interpretation of SUBATTRIBUTESET if neither conjunction nor disjunction set is present

$attributeSet == subAttributeSet \times conjunctionOrDisjunctionAttributeSet[0\ ..\ 1]$
$[attributeSet']$

$conjunctionOrDisjunctionAttributeSet ::=$
    $attset\_conjattset \langle\!\langle conjunctionAttributeSet \rangle\!\rangle \mid$
    $attset\_disjattset \langle\!\langle disjunctionAttributeSet \rangle\!\rangle$

$i\_attributeSet :$
$\quad Substrate \rightarrow attributeSet \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ attset : attributeSet \bullet$
$i\_attributeSet\ ss\ attset =$
$(\textbf{let}\ lhs == i\_subAttributeSet\ ss\ (first\ attset);\ rhs == second\ attset \bullet$
$\textbf{if}\ \#rhs = 0\ \textbf{then}$
$\quad lhs$
$\textbf{else if}\ head\ rhs \in \mathrm{ran}\ attset\_conjattset\ \textbf{then}$
$\quad intersect\ lhs\ (i\_conjunctionAttributeSet\ ss\ (attset\_conjattset{\sim}(head\ rhs)))$
$\textbf{else}$
$\quad union\ lhs\ (i\_disjunctionAttributeSet\ ss\ (attset\_conjattset{\sim}(head\ rhs))))$

$i\_attributeSet' :$
$\quad Substrate \rightarrow attributeSet' \rightarrow Sctids\_or\_Error$

3.4.1 conjunctionAttributeSet

> *A conjunction attribute set consists of one or more conjunction operators, each followed by a subAttributeSet.*

> conjunctionAttributeSet = 1*(ws conjunction ws subAttributeSet)

The interpretation of a CONJUNCTIONATTRIBUTESET is the <u>intersection</u> of the interpretations of its SUBATTRIBUTESETS .

$conjunctionAttributeSet == \mathrm{seq}_1\ subAttributeSet$

$i\_conjunctionAttributeSet :$
$\quad Substrate \rightarrow conjunctionAttributeSet \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ conjset : conjunctionAttributeSet \bullet$
$i\_conjunctionAttributeSet\ ss\ conjset =$
$\textbf{if}\ tail\ conjset = \langle\rangle\ \textbf{then}$
$\quad i\_subAttributeSet\ ss\ (head\ conjset)$
$\textbf{else}$
$\quad intersect\ (i\_subAttributeSet\ ss\ (head\ conjset))\ (i\_conjunctionAttributeSet\ ss\ (tail\ conjset))$

3.4.2 disjunctionAttributeSet

> *A disjunction attribute set consists of one or more disjunction operators, each followed by a subAttributeSet.*

The interpretation of a DISJUNCTIONATTRIBUTESET is the <u>union</u> of the interpretations of its SUBATTRIBUTESETS.

$$disjunctionAttributeSet == \mathrm{seq}_1\ subAttributeSet$$

$i\_disjunctionAttributeSet :$
$\quad Substrate \to disjunctionAttributeSet \to Sctids\_or\_Error$

$\forall\, ss : Substrate;\ disjset : disjunctionAttributeSet \bullet$
$i\_disjunctionAttributeSet\ ss\ disjset =$
if $tail\ disjset = \langle\rangle$ **then**
$\quad i\_subAttributeSet\ ss\ (head\ disjset)$
else
$\quad union\ (i\_subAttributeSet\ ss\ (head\ disjset))\ (i\_disjunctionAttributeSet\ ss\ (tail\ disjset))$

3.4.3 subAttributeSet

A subAttributeSet is either an attribute or a bracketed attribute set.

The interpretation of a SUBATTRIBUTESET is either the interpretation of the ATTRIBUTE or the interpretation of the ATTRIBUTESET.

$subAttributeSet ::=$
$\quad subaset\_attribute \langle\!\langle attribute \rangle\!\rangle\ |$
$\quad subaset\_attset \langle\!\langle attributeSet' \rangle\!\rangle$

$i\_subAttributeSet :$
$\qquad Substrate \to subAttributeSet \to Sctids\_or\_Error$

$\forall\, ss : Substrate;\ subaset : subAttributeSet \bullet$
$i\_subAttributeSet\ ss\ subaset =$
if $subaset \in \mathrm{ran}\ subaset\_attribute$ **then**
$\quad i\_attribute\ ss\ (subaset\_attribute^{\sim} subaset)$
else
$\quad i\_attributeSet'\ ss\ (subaset\_attset^{\sim} subaset)$

3.5 attribute

> *An attribute is a name-value pair expressing a single refinement of the containing expression constraint. Either the attribute value must satisfy (or not) the given expression constraint, the attribute value is compared with a given numeric value (integer or decimal) using a numeric comparison operator, or the attribute value must be equal to (or not equal to) the given string value. The attribute may optionally be preceded by a cardinality constraint, a reverse flag and/or an attribute operator.*

---

attribute = [cardinality ws] [reverseFlag ws] [attributeOperator ws] attributeName ws (expressionComparisonOperator ws expressionConstraintValue / numericComparisonOperator ws numericValue / stringComparisonOperator ws stringValue   numericValue = "#" (decimalValue / integerValue)

Conceptually, the interpretation of ATTRIBUTE   consists of the following steps:

1. Interpret ATTRIBUTENAME
2. Interpret ATTRIBUTEOPERATOR   against the result of the previous step
3. Interpret one of:
   - The EXPRESSIONCOMPARISONOPERATOR   applied to the REVERSEFLAG , the interpretation of ATTRIBUTEOPERATOR   and EXPRESSIONCONSTRAINTVALUE .
   - The NUMERICCOMPARISONOPERATOR   applied to the interpretation of ATTRIBUTEOPERATOR   and NUMERICVALUE .
   - The STRINGCOMPARISONOPERATOR   applied to the interpretation of ATTRIBUTEOPERATOR   and STRINGVALUE .
4. Interpret CARDINALITY   against the result of the previous step

Note that the REVERSEFLAG   only applies in the case of EXPRESSIONCOMPARISONOPERATOR . Not sure what to do if it is present in the other two cases...

The schema below provides names the various components of an attribute.

$\underline{\text{attribute}}$
$card : cardinality[0 \ldots 1]$
$rf : reverseFlag[0 \ldots 1]$
$attOper : constraintOperator[0 \ldots 1]$
$name : attributeName$
$opValue : attributeOperatorValue$

$attributeOperatorValue ::=$
$\quad attrib\_expr \langle\!\langle expressionComparisonOperator \times expressionConstraintValue \rangle\!\rangle \mid$
$\quad attrib\_num \langle\!\langle numericComparisonOperator \times numericValue \rangle\!\rangle \mid$
$\quad attrib\_str \langle\!\langle stringComparisonOperator \times stringValue \rangle\!\rangle$
$numericValue ::= nv\_decimal \langle\!\langle decimalValue \rangle\!\rangle \mid nv\_integer \langle\!\langle \mathbb{N} \rangle\!\rangle$
$[reverseFlag]$

22

$i\_attribute :$
$\quad Substrate \rightarrow attribute \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ att : attribute \bullet$
$i\_attribute\ ss\ att =$
$(\textbf{let}\ att\_sctids == i\_attributeOperator\ ss\ att.attOper\ (ss.i\_attributeName\ att.name) \bullet$
$\textbf{if}\ att\_sctids \in \mathrm{ran}\ error\ \textbf{then}$
$\quad att\_sctids$

$\textbf{else if}\ att.opValue \in \mathrm{ran}\ attrib\_expr\ \textbf{then}$
$\quad (\textbf{let}\ expr\_interp ==$
$\quad i\_expressionComparisonOperator\ ss\ att.rf\ (result\_sctids\ att\_sctids)\ (attrib\_expr^{\sim} att.opValue) \bullet$
$\quad\quad i\_att\_cardinality\ ss\ att.card\ expr\_interp)$

$\textbf{else if}\ att.opValue \in \mathrm{ran}\ attrib\_num$
$\quad \textbf{then}$
$\quad (\textbf{let}\ num\_interp ==$
$\quad i\_numericComparisonOperator\ ss\ att.rf\ (result\_sctids\ att\_sctids)\ (attrib\_num^{\sim} att.opValue) \bullet$
$\quad\quad i\_att\_cardinality\ ss\ att.card\ num\_interp)$

$\textbf{else}$
$\quad (\textbf{let}\ str\_interp ==$
$\quad i\_stringComparisonOperator\ ss\ att.rf\ (result\_sctids\ att\_sctids)\ (attrib\_str^{\sim} att.opValue) \bullet$
$\quad\quad i\_att\_cardinality\ ss\ att.card\ str\_interp))$

3.5.1 attribute cardinality

> *The cardinality of an attribute represents a constraint on the minimum and maximum number of instances of the given attribute on each concept. The cardinality is enclosed in square brackets with the minimum cardinality appearing first, followed by two dots and then the maximum cardinality. The minimum cardinality must always be less than or equal to the maximum cardinality. A maximum cardinality of 'many' indicates that there is no limit on the number of times the attribute may appear on each concept.*

> cardinality = "[" nonNegativeIntegerValue to (nonNegativeIntegerValue / many) "]"

$unlimitedNat ::= num\langle\!\langle \mathbb{N} \rangle\!\rangle \mid many$

$\_\,cardinality _____$
$min : \mathbb{N};\ max : unlimitedNat$

The interpretation of $optional[cardinality]$ in the context of an attribute is:

1. *error* if there is an error in the supplied list of quads

2. *i_required_cardinality* 1 *many* if the cardinality isn't supplied
3. *i_required_cardinality cardinality* if the minimum cardinality is > 0
4. *i_optionali_cardinality ss cardinality* if the minimum cardinality $= 0$

$\rule{0pt}{0pt}$

> *i_att_cardinality* :
> $Substrate \rightarrow cardinality[0 \mathbin{..} 1] \rightarrow Quads\_or\_Error \rightarrow Sctids\_or\_Error$
> ___
> $\forall ss : Substrate;\ ocard : cardinality[0 \mathbin{..} 1];\ qore : Quads\_or\_Error \bullet$
> *i_att_cardinality ss ocard qore* =
> **if** *qore* $\in$ ran *qerror* **then**
>   *error*(*qerror*$^{\sim}$ *qore*)
> **else if** $\# ocard = 0$ **then**
>   *ok* (*i_required_cardinality* 1 *many qore*)
> **else** (**let** *card* == *head ocard* $\bullet$
>   **if** *card.min* > 0 **then**
>   *ok* (*i_required_cardinality card.min card.max qore*)
>   **else**
>   *ok* (*i_optional_cardinality ss card.max qore*))

i_required_cardinality returns the set of subject or target sctids that meet
the cardinality requirements

> *i_required_cardinality* :
> $\mathbb{N} \rightarrow unlimitedNat \rightarrow Quads\_or\_Error\ \rightarrow \mathbb{P}\,sctId$
> ___
> $\forall min : \mathbb{N};\ max : unlimitedNat;\ qore : Quads\_or\_Error \bullet$
> *i_required_cardinality min max qore* =
> **if** *quad_direction qore* = *source_direction* **then**
> $\{subj : \{q : quads\_for\ qore \bullet q.s\}\ |$
> $evalCardinality[Quad]\ min\ max\ \{q : quads\_for\ qore\ |\ q.s = subj\} \neq \emptyset\}$
> **else**
> $\{targ : \{q : quads\_for\ qore \bullet t\_sctid^{\sim} q.t\}\ |$
> $evalCardinality[Quad]\ min\ max\ \{q : quads\_for\ qore\ |\ (t\_sctid^{\sim} q.t) = targ\} \neq \emptyset\}$

i_optional_cardinality returns all of the concepts in the substrate that don't
fail the cardinality test. The failing concepts are equivalent to the set of all
subject (or target) concepts in the set of quads minus the set of concepts that
pass.

> *i_optional_cardinality* :
> $Substrate \rightarrow unlimitedNat \rightarrow Quads\_or\_Error\ \rightarrow \mathbb{P}\,sctId$
> ___
> $\forall ss : Substrate;\ max : unlimitedNat;\ qore : Quads\_or\_Error \bullet$
> *i_optional_cardinality ss max qore* =
> **if** *quad_direction qore* = *source_direction* **then**
> $ss.concepts \setminus (\{q : quads\_for\ qore \bullet q.s\} \setminus (i\_required\_cardinality\ 0\ max\ qore))$
> **else**
> $ss.concepts \setminus (\{q : quads\_for\ qore \bullet t\_sctid^{\sim} q.t\} \setminus (i\_required\_cardinality\ 0\ max\ qore))$

3.5.2 expressionConstraintValue

> *An expression constraint value is either a simple expression constraint, or is a refined or compound expression constraint enclosed in brackets.*

expressionConstraintValue = simpleExpressionConstraint / "(" ws (refinedExpressionConstraint / compoundExpressionConstraint) ws ")"

The interpretation of ExpressionConstraintValue is the interpretation of the corresponding simpleExpressionConstraint, refinedExpressionConstraint or compoundExpressionConstraint

$$expressionConstraintValue ::=$$
$$expression\_simple \langle\!\langle simpleExpressionConstraint \rangle\!\rangle \mid$$
$$expression\_refined \langle\!\langle refinedExpressionConstraint' \rangle\!\rangle \mid$$
$$expression\_compound \langle\!\langle compoundExpressionConstraint \rangle\!\rangle$$

$$i\_expressionConstraintValue :$$
$$\quad Substrate \rightarrow expressionConstraintValue \rightarrow Sctids\_or\_Error$$

$$\forall\, ss : Substrate;\ ecv : expressionConstraintValue \bullet$$
$$i\_expressionConstraintValue\ ss\ ecv =$$
if $ecv \in \mathrm{ran}\ expression\_simple$ **then**
$$\quad i\_simpleExpressionConstraint\ ss\ (expression\_simple^\sim ecv)$$
else if $ecv \in \mathrm{ran}\ expression\_refined$ **then**
$$\quad i\_refinedExpressionConstraint'\ ss\ (expression\_refined^\sim ecv)$$
else
$$\quad i\_compoundExpressionConstraint\ ss\ (expression\_compound^\sim ecv)$$

3.6 attributeGroup

> *An attribute group contains a collection of attributes that operate together as part of the refinement of the containing expression constraint. An attribute group may optionally be preceded by a cardinality. An attribute group cardinality indicates the minimum and maximum number of attribute groups that must satisfy the given attributeSet constraint for the expression constraint to be satisfied.*

attributeGroup = [cardinality ws] "{" ws attributeSet ws "}"

$$attributeGroup == cardinality[0\,..\,1] \times attributeSet$$

The difference between the interpretation of an ATTRIBUTEGROUP and an ATTRIBUTESET is that, within an ATTRIBUTEGROUP all of the qualifying SCTIDs must belong to the same group [1]. Outside, a SCTID would qualify an conjunction if one group passed the first match and a different one the second. *Within a group*, however, conjunctions and disjunctions only count if they both apply in the same group. This means that we have to re-interpret the meaning of ATTRIBUTESET on down in terms of *IDGroup*s rather than *Quads*

The interpretation of an attribute group is the application of a cardinality constraint to all of the *sctId*s that pass the grouped attribute set.

$i\_attributeGroup : Substrate \rightarrow attributeGroup \rightarrow Sctids\_or\_Error$

$\forall ss : Substrate; \ ag : attributeGroup \bullet$
$i\_attributeGroup \ ss \ ag =$
$\quad i\_group\_cardinality \ ss \ (first \ ag) \ (i\_groupAttributeSet \ ss \ (second \ ag))$

3.6.1 attribute group cardinality

The interpretation of CARDINALITY within the context of an attribute group is as follows:

- If an error has been encountered anywhere in the process, it should be propagated.
- If the cardinality is omitted, it is interpreted as [1 .. *]
- If the minimum cardinality is >0 then return the set of sctids with that appear at least *min* and at most *max* groups
- If the minimum cardinality is 0 then return every sctid that doesn't fail the max cardinality

$i\_group\_cardinality :$
$\quad Substrate \rightarrow cardinality[0 \, . \, . \, 1] \rightarrow sctIdGroups\_or\_Error \rightarrow Sctids\_or\_Error$

$\forall ss : Substrate; \ ocard : cardinality[0 \, . \, . \, 1]; \ idg : sctIdGroups\_or\_Error \bullet$
$i\_group\_cardinality \ ss \ ocard \ idg =$
if $idg \in \text{ran} \ gerror$ **then**
$\quad error(gerror^\sim idg)$

else (**let** $gvals == group\_value^\sim idg \bullet$
$\quad$ **if** $\#ocard = 0$ **then**
$\quad\quad ok(i\_required\_group\_cardinality \ 1 \ many \ gvals)$

$\quad$ **else** (**let** $card == head \ ocard \bullet$
$\quad\quad$ **if** $card.min > 0$ **then**
$\quad\quad\quad ok(i\_required\_group\_cardinality \ card.min \ card.max \ gvals)$

$\quad\quad$ **else**
$\quad\quad\quad ok(i\_optional\_group\_cardinality \ ss \ card.max \ gvals)))$

---

[1]The zero group is a special case, where each quad in a zero group is viewed as its own unique group

i_required_group_cardinality returns the set *sctids* that pass in at least *min* groups and at most *max* groups

> *i_required_group_cardinality* :
> $\mathbb{N} \to unlimitedNat \to \mathbb{P}\, sctIdGroup \to \mathbb{P}\, sctId$
> ___
> $\forall\, min : \mathbb{N};\; max : unlimitedNat;\; groups : \mathbb{P}\, sctIdGroup \bullet$
> *i_required_group_cardinality min max groups* =
> $\{sctid : \{s : groups \bullet first\, s\}\,|$
> $evalCardinality[sctIdGroup]\, min\, max\{g : groups\,|\, first\, g = sctid\} \neq \emptyset\}$

i_optional_group_cardinality returns the set of *sctids* that pass in at most *max* groups, which is equivalent to the set of all possible *sctIds* (i.e. *ss.concepts*) minus the set of *sctids* that pass in more than *max* groups.

> *i_optional_group_cardinality* :
> $Substrate \to unlimitedNat \to \mathbb{P}\, sctIdGroup \to \mathbb{P}\, sctId$
> ___
> $\forall\, ss : Substrate;\; max : unlimitedNat;\; groups : \mathbb{P}\, sctIdGroup \bullet$
> *i_optional_group_cardinality ss max groups* =
> $ss.concepts\backslash$
> $(\{s : groups \bullet first\, s\} \setminus (i\_required\_group\_cardinality\, 0\, max\, groups))$

3.6.2 attributeSet inside a groups

The interpretation of an ATTRIBUTESET a group is the interpretation of the SUBATTRIBUTESET optionally intersected or union with additional conjunction or disjunction attribute sets.

> *i_groupAttributeSet* :
> $Substrate \to attributeSet \to sctIdGroups\_or\_Error$
> ___
> $\forall\, ss : Substrate;\; attset : attributeSet \bullet$
> *i_groupAttributeSet ss attset* =
> (**let** *lhs* == *i_groupSubAttributeSet ss* (*first attset*); *rhs* == *second attset* $\bullet$
> **if** $\#rhs = 0$ **then**
>   *lhs*
> **else if** *head rhs* $\in$ ran *attset_conjattset* **then**
>   *gintersect lhs* (*i_groupConjunctionAttributeSet ss* (*attset_conjattset*$^\sim$(*head rhs*)))
> **else**
>   *gunion lhs* (*i_groupDisjunctionAttributeSet ss* (*attset_conjattset*$^\sim$(*head rhs*))))

> *i_groupAttributeSet'* :
> $Substrate \to attributeSet' \to sctIdGroups\_or\_Error$

3.6.3 conjunctionAttributeSet inside a group

Apply the intersect operator to the interpretation of each subAttributeSet in a group context. This is the same as the non-grouped *subAttributeSet* with the

exception that the inputs to and result of the function is *sctId / groupId* tuples rather than

$i\_groupConjunctionAttributeSet :$
  $Substrate \rightarrow conjunctionAttributeSet \rightarrow sctIdGroups\_or\_Error$

$\forall ss : Substrate;\ conjset : conjunctionAttributeSet \bullet$
$i\_groupConjunctionAttributeSet\ ss\ conjset =$
if $tail\ conjset = \langle\rangle$ **then**
  $i\_groupSubAttributeSet\ ss\ (head\ conjset)$
else
  $gintersect\ (i\_groupSubAttributeSet\ ss\ (head\ conjset))\ (i\_groupConjunctionAttributeSet\ ss\ (tail\ conj$

3.6.4 disjunctionAttributeSet inside a group

Apply the union operator to the interpretation of each subAttributeSet in a group context. This is the same as the non-grouped *subAttributeSet* with the exception that the inputs to and result of the function is *sctId / groupId* tuples rather than just *sctIds*.

$i\_groupDisjunctionAttributeSet :$
  $Substrate \rightarrow disjunctionAttributeSet \rightarrow sctIdGroups\_or\_Error$

$\forall ss : Substrate;\ disjset : disjunctionAttributeSet \bullet$
$i\_groupConjunctionAttributeSet\ ss\ disjset =$
if $tail\ disjset = \langle\rangle$ **then**
  $i\_groupSubAttributeSet\ ss\ (head\ disjset)$
else
  $gunion\ (i\_groupSubAttributeSet\ ss\ (head\ disjset))\ (i\_groupConjunctionAttributeSet\ ss\ (tail\ disjset)$

3.6.5 subAttributeSet inside a group

This is the same as the non-grouped *subAttributeSet* with the exception that the result of the function is *sctId / groupId* tuples rather than just *sctIds*.

$i\_groupSubAttributeSet :$
  $Substrate \rightarrow subAttributeSet \rightarrow sctIdGroups\_or\_Error$

$\forall ss : Substrate;\ subaset : subAttributeSet \bullet$
$i\_groupSubAttributeSet\ ss\ subaset =$
if $subaset \in \mathrm{ran}\ subaset\_attribute$
  then $i\_groupAttribute\ ss\ (subaset\_attribute^\sim subaset)$
else $i\_groupAttributeSet'\ ss\ (subaset\_attset^\sim subaset)$

3.6.6 attribute inside a group

The interpretation of an attribute within the context of a group is the set of *sctIds* and the group or groups in which they were valid. It is possible for the same *sctId* to valid in the interpretation of a given attribute for more than one

group. The union and intersection operators operate on the *sctId* / *groupId* tuples (ne. *sctIdGroup*).

$$\begin{array}{l}
\textit{i\_groupAttribute} : \\
\quad \textit{Substrate} \rightarrow \textit{attribute} \rightarrow \textit{sctIdGroups\_or\_Error} \\
\hline
\forall\, ss : \textit{Substrate};\ \ att : \textit{attribute} \bullet \\
\textit{i\_groupAttribute}\ ss\ att = \\
(\textbf{let}\ \textit{att\_sctids} == \textit{i\_attributeOperator}\ ss\ att.attOper\ (ss.\textit{i\_attributeName}\ att.name) \bullet \\
\textbf{if}\ \textit{att\_sctids} \in \mathrm{ran}\ \textit{error} \\
\quad \textbf{then}\ \textit{gerror}(\textit{error}^{\sim}\ \textit{att\_sctids}) \\[4pt]
\textbf{else if}\ att.opValue \in \mathrm{ran}\ \textit{attrib\_expr}\ \textbf{then} \\
\quad (\textbf{let}\ \textit{expr\_interp} == \\
\quad \textit{i\_expressionComparisonOperator}\ ss\ att.rf\ (\textit{result\_sctids}\ \textit{att\_sctids})\ (\textit{attrib\_expr}^{\sim}\ att.opValue) \bullet \\
\qquad \textit{i\_att\_group\_cardinality}\ ss\ att.card\ \textit{expr\_interp}) \\[4pt]
\textbf{else if}\ att.opValue \in \mathrm{ran}\ \textit{attrib\_num}\ \textbf{then} \\
\quad (\textbf{let}\ \textit{num\_interp} == \\
\quad \textit{i\_numericComparisonOperator}\ ss\ att.rf\ (\textit{result\_sctids}\ \textit{att\_sctids})\ (\textit{attrib\_num}^{\sim}\ att.opValue) \bullet \\
\qquad \textit{i\_att\_group\_cardinality}\ ss\ att.card\ \textit{num\_interp}) \\[4pt]
\textbf{else} \\
\quad (\textbf{let}\ \textit{str\_interp} == \\
\quad \textit{i\_stringComparisonOperator}\ ss\ att.rf\ (\textit{result\_sctids}\ \textit{att\_sctids})\ (\textit{attrib\_str}^{\sim}\ att.opValue) \bullet \\
\qquad \textit{i\_att\_group\_cardinality}\ ss\ att.card\ \textit{str\_interp}))
\end{array}$$

3.6.7 cardinality inside a group

Cardinality is interpreted differently within the context of a a group. In particular:

- Zero groups are treated as "singletons" – every quad with a zero group is treated as a group unto itself.
- A sctid can pass in the context of more than one group. Intersections, unions, etc. apply to sctid's in the *same* group.

i_att_group_cardinality takes an optional cardinality and a set of quads and returns a list sctids and the corresponding groups in which they passed the cardinality constraint. Note that, in the case of the zero group, the actual *Quad* that passed is associated with the sctid rather than the group identifier itself.

$i\_att\_group\_cardinality :$
$\quad Substrate \to cardinality[0 \mathbin{..} 1] \to Quads\_or\_Error \to sctIdGroups\_or\_Error$

$\forall ss : Substrate; \ ocard : cardinality[0 \mathbin{..} 1]; \ qore : Quads\_or\_Error \bullet$
$i\_att\_group\_cardinality \ ss \ ocard \ qore =$
if $qore \in \mathrm{ran}\ qerror$ **then**
$\quad gerror(qerror^{\sim} qore)$

else if $\#ocard = 0$ **then**
$\quad group\_value\ (i\_required\_att\_group\_cardinality\ 1\ many\ qore)$

else (**let** $card == head\ ocard \bullet$
$\quad$ **if** $card.min > 0$ **then**
$\quad group\_value\ (i\_required\_att\_group\_cardinality\ card.min\ card.max\ qore)$

else
$\quad group\_value\ (i\_optional\_att\_group\_cardinality\ ss\ card.max\ qore))$

$i\_required\_att\_group\_cardinality$ returns the set of subject or target sctids combined with the group(s) (or $Quads$) in which they qualified

$i\_required\_att\_group\_cardinality :$
$\quad \mathbb{N} \to unlimitedNat \to Quads\_or\_Error \ \to \mathbb{P}\ sctIdGroup$

$\forall min : \mathbb{N}; \ max : unlimitedNat; \ qore : Quads\_or\_Error \bullet$
$i\_required\_att\_group\_cardinality\ min\ max\ qore =$
if $quad\_direction\ qore = source\_direction$ **then**
$\quad \{subjgroup : \{q : quads\_for\ qore \mid q.g \neq zero\_group \bullet (q.s, quadGroup\ q)\} \mid$
$\quad\quad evalCardinality[Quad]\ min\ max\ \{q : quads\_for\ qore \mid$
$\quad\quad\quad q.s = first\ subjgroup \wedge q.g = (ug^{\sim}(second\ subjgroup))\} \neq \emptyset\}$
$\quad\quad\quad\quad \cup$
$\quad \{q : quads\_for\ qore \mid q.g = zero\_group \wedge$
$\quad\quad evalCardinality[Quad]\ min\ max\ \{q\} \neq \emptyset \bullet (q.s, quadGroup\ q)\}$

else
$\quad \{targgroup : \{q : quads\_for\ qore \mid$
$\quad\quad q.g \neq zero\_group \wedge q.t \in \mathrm{ran}\ t\_sctid \bullet (t\_sctid^{\sim} q.t, quadGroup\ q)\} \mid$
$\quad\quad evalCardinality[Quad]\ min\ max\ \{q : quads\_for\ qore \mid$
$\quad\quad\quad t\_sctid^{\sim} q.t = first\ targgroup \wedge$
$\quad\quad\quad q.g = (ug^{\sim}(second\ targgroup))\} \neq \emptyset\}$
$\quad\quad\quad\quad \cup$
$\quad \{q : quads\_for\ qore \mid q.g = zero\_group \wedge$
$\quad\quad evalCardinality[Quad]\ min\ max\ \{q\} \neq \emptyset \bullet (t\_sctid^{\sim} q.t, quadGroup\ q)\}$

$i\_optional\_att\_group\_cardinality$ returns the set of subject or target sctids along with all of the corresponding groups that *fail* the cardinality test. The failing list is equivalent to the set of all possible sctid / non-zero groups in the substrate minus the set of sctid / non-zero groups that fail the maximum cardinality. The set of failures is determined by removing the set of passing tuples from the total possible set in the supplied quads.

$i\_optional\_att\_group\_cardinality :$
$\quad Substrate \rightarrow unlimitedNat \rightarrow Quads\_or\_Error \rightarrow \mathbb{P}\, sctIdGroup$

$\forall\, ss : Substrate;\ max : unlimitedNat;\ qore : Quads\_or\_Error \bullet$
$i\_optional\_att\_group\_cardinality\ ss\ max\ qore =$
if $quad\_direction\ qore = source\_direction$ **then**
$\quad (\{rel : ss.relationships \bullet (rel.s, quadGroup\ rel)\} \setminus$
$\quad \{q : quads\_for\ qore \bullet (q.s, quadGroup\ q)\}) \cup$
$\qquad i\_required\_att\_group\_cardinality\ 0\ max\ qore$

else
$\quad (\{rel : ss.relationships \mid rel.t \in \mathrm{ran}\ t\_sctid \bullet (t\_sctid^{\sim} rel.t, quadGroup\ rel)\} \setminus$
$\quad \{q : quads\_for\ qore \mid q.g \neq zero\_group \bullet (t\_sctid^{\sim} q.t, quadGroup\ q)\}) \cup$
$\qquad i\_required\_att\_group\_cardinality\ 0\ max\ qore$

4 Substrate Interpretations

This section defines the interpretations that are realized against the substrate.

4.1 constraintOperator

> *A constraint operator is either 'descendantOrSelfOf', 'descendantOf', 'ancestorOrSelfOf', or 'ancestorOf'.*

constraintOperator = descendantOrSelfOf / descendantOf / ancestorOrSelfOf / ancestorOf

The interpretation of *constraintOperator* [0 . . 1] is one of:

- The input if an operator isn't supplied or if it contains an error
- The union of descendants of all members of the input, as provided by the substrate if the operator is *descendantOf*
- The union of descendants plus the input if the operator is *descendantOrSelfOf*
- The union of ancestors of all members of the input, as provided by the substrate if the operator is *ancestorOf*
- The union of ancestors plus the input if the operator is *ancestorsOrSelfOf*

Note that it is possible for some or all of the members of the input to be included in the *descendantOf* or *ancestorOf* operations as the descendants of one input sctid may include another input sctid.

The *completeFun* function assures that an operation on any identifier that isn't in the domain of the substrate *descendants* or *ancestors* function returns the empty set ($\emptyset$).

$constraintOperator ::=$
$\qquad descendantOrSelfOf \mid descendantOf \mid ancestorOrSelfOf \mid ancestorOf$

$i\_constraintOperator :$
$\quad Substrate \rightarrow constraintOperator[0\,..\,1] \rightarrow Sctids\_or\_Error \rightarrow Sctids\_or\_Error$

$completeFun : (sctId \nrightarrow \mathbb{P}\,sctId) \rightarrow sctId \rightarrow \mathbb{P}\,sctId$

$\forall\,ss : Substrate;\ oco : constraintOperator[0\,..\,1];\ input : Sctids\_or\_Error \bullet$
$i\_constraintOperator\ ss\ oco\ input =$
$\quad \textbf{if}\ error^\sim input \in ERROR \vee \#oco = 0\ \textbf{then}$
$\qquad input$
$\quad \textbf{else if}\ head\ oco = descendantOrSelfOf\ \textbf{then}$
$\qquad ok(\bigcup\{id : result\_sctids\ input \bullet$
$\qquad\qquad completeFun\ ss.descendants\ id\} \cup result\_sctids\ input)$
$\quad \textbf{else if}\ head\ oco = descendantOf\ \textbf{then}$
$\qquad ok(\bigcup\{id : result\_sctids\ input \bullet$
$\qquad\qquad completeFun\ ss.descendants\ id\ \})$
$\quad \textbf{else if}\ head\ oco = ancestorOrSelfOf\ \textbf{then}$
$\qquad ok(\bigcup\{id : result\_sctids\ input \bullet$
$\qquad\qquad completeFun\ ss.ancestors\ id\} \cup result\_sctids\ input)$
$\quad \textbf{else}$
$\qquad ok(\bigcup\{id : result\_sctids\ input$
$\qquad\qquad \bullet completeFun\ ss.ancestors\ id\ \})$

$\forall\,f : (sctId \nrightarrow \mathbb{P}\,sctId);\ id : sctId \bullet completeFun\ f\ id =$
$\quad \textbf{if}\ id \in \mathrm{dom}\,f\ \textbf{then}\ f\ id\ \textbf{else}\ \emptyset$

4.2   attributeOperator

An attribute operator indicates that instead of just matching the named attribute with the given attribute value, any descendants of or any descendants or self of the named attribute may match the given attribute value.

attributeOperator = descendantOrSelfOf / descendantOf

The interpretation of *attributeOperator* is the same as the interpretation of *constraintOperator*

$attributeOperator == constraintOperator$
$i\_attributeOperator == i\_constraintOperator$

4.3   expressionComparisonOperator

Attributes whose value is a concept may be compared to an expression constraint using either equals ("=") or not equals ("!="). In the full syntax "¡¿" and "not =" (case insensitive) are also valid ways to represent not equals.

The interpretation of $\textsf{EXPRESSIONCOMPARISONOPERATOR}$ is the set of all quads in the substrate having a target (or subject if the direction is reverse) that are (or are not) in the interpretation of the $\textsf{EXPRESSIONCONSTRAINTVALUE}$

$$expressionComparisonOperator ::= eco\_eq \mid eco\_neq$$

$i\_expressionComparisonOperator :$
 $Substrate \rightarrow reverseFlag[0\,.\,.\,1] \rightarrow \mathbb{P}\,sctId \rightarrow$
 $(expressionComparisonOperator \times expressionConstraintValue) \rightarrow Quads\_or\_Error$

$\forall\,ss : Substrate;\ rf : reverseFlag[0\,.\,.\,1];\ atts : \mathbb{P}\,sctId;$
 $ec : (expressionComparisonOperator \times expressionConstraintValue) \bullet$
$i\_expressionComparisonOperator\ ss\ rf\ atts\ ec =$
$(\textbf{let}\ ecv == i\_expressionConstraintValue\ ss\ (second\ ec) \bullet$
 $\textbf{if}\ ecv \in \text{ran}\ error\ \textbf{then}$
  $qerror(error^{\sim} ecv)$
 $\textbf{else if}\ \#reverseFlag = 0 \wedge (first\ ec = eco\_eq)\ \textbf{then}$
  $quad\_value(\{q : ss.relationships \mid$
   $q.t \in \text{ran}\ t\_sctid \wedge (t\_sctid^{\sim} q.t) \in (result\_sctids\ ecv)\}, source\_direction)$
 $\textbf{else if}\ \#reverseFlag > 0 \wedge (first\ ec = eco\_eq)\ \textbf{then}$
  $quad\_value(\{q : ss.relationships \mid$
   $q.s \in (result\_sctids\ ecv)\}, targets\_direction)$
 $\textbf{else if}\ \#reverseFlag = 0 \wedge (first\ ec = eco\_neq)\ \textbf{then}$
  $quad\_value(\{q : ss.relationships \mid$
   $q.t \in \text{ran}\ t\_sctid \wedge (t\_sctid^{\sim} q.t) \notin (result\_sctids\ ecv)\}, source\_direction)$
 $\textbf{else}$
  $quad\_value(\{q : ss.relationships \mid$
   $q.s \notin (result\_sctids\ ecv)\}, targets\_direction))$

4.4   numericComparisonOperator

Attributes whose value is numeric (i.e. integer or decimal) may be compared to a specific concrete value using a variety of comparison operators, including equals ("="), less than ("<"), less than or equals ("<="), greater than (">"), greater than or equals (">=") and not equals ("!="). In the full syntax "<>" and "not =" (case insensitive) are also valid ways to represent not equals.

The interpretation of $\textsf{NUMERICCOMPARISONOPERATOR}$ is the set of all quads in the substrate having a *concreteValue* target of type *cv_integer* or *cv_decimal* that

(or do not meet) the supplied criteria. As concrete values cannot occur as subjects, this function returns the empty set if the reverse flag is supplied

$$numericComparisonOperator ::= nco\_eq \mid nco\_neq \mid nco\_gt \mid nco\_ge \mid nco\_lt \mid nco\_le$$

$$
\begin{array}{l}
i\_numericComparisonOperator : \\
\quad Substrate \rightarrow reverseFlag[0 \mathinner{\ldotp\ldotp} 1] \rightarrow \mathbb{P}\, sctId \rightarrow \\
\quad (numericComparisonOperator \times numericValue) \rightarrow Quads\_or\_Error \\
\hline
\forall\, ss : Substrate;\ rf : reverseFlag[0 \mathinner{\ldotp\ldotp} 1];\ atts : \mathbb{P}\, sctId; \\
\quad ncv : (numericComparisonOperator \times numericValue) \bullet \\
i\_numericComparisonOperator\ ss\ rf\ atts\ ncv = \\
\textbf{if}\ \#rf > 0\ \textbf{then} \\
\quad quad\_value(\emptyset, targets\_direction) \\
\textbf{else} \\
\quad quad\_value(\{q : ss.relationships \mid q.a \in atts \land q.t \in \operatorname{ran} t\_concrete\ \land \\
numericComparison\,(t\_concrete^{\sim}q.t)\,(first\ ncv) = (second\ ncv)\}, source\_direction)
\end{array}
$$

Numeric comparison function. Compares a *concreteValue* with a *numericValue*.

TODO: Need to specify the rules for comparing integers and decimal numbers as well as decimal/decimal. As an example, does "5.00" = "5.0"

$$
\begin{array}{l}
numericComparison : \\
\quad concreteValue \rightarrow numericComparisonOperator \nrightarrow numericValue
\end{array}
$$

4.5   stringComparisonOperator

Attributes whose value is numeric may be compared to an expression constraint using either equals ("=") or not equals ("!="). In the full syntax "¡¿" and "not =" (case insensitive) are also valid ways to represent not equals.

stringComparisonOperator = "=" / "!="

$$stringComparisonOperator ::= sco\_eq \mid sco\_neq$$

$$i\_stringComparisonOperator:$$
$$Substrate \to reverseFlag[0\,..\,1] \to \mathbb{P}\,sctId \to$$
$$(stringComparisonOperator \times stringValue) \to Quads\_or\_Error$$

$$\forall\, ss : Substrate;\ rf : reverseFlag[0\,..\,1];\ atts : \mathbb{P}\,sctId;$$
$$scv : (stringComparisonOperator \times stringValue) \bullet$$
$$i\_stringComparisonOperator\ ss\ rf\ atts\ scv =$$
if $\#rf > 0$ **then**
$$quad\_value(\emptyset, targets\_direction)$$
else
$$quad\_value(\{\,q : ss.relationships \mid q.a \in atts \land q.t \in \mathrm{ran}\,t\_concrete \land$$
$$stringComparison\,(t\_concrete^\sim q.t)\,(first\ scv) = (second\ scv)\}, source\_direction)$$

String comparison function. Compares a *concreteValue* with a *stringValue*.

$$stringComparison:$$
$$concreteValue \to stringComparisonOperator \nrightarrow stringValue$$

4.6   focusConcept

A focus concept is a concept reference or wild card, which is optionally preceded by a member of function. A memberOf function should be used only when the conceptReference refers to a reference set concept, or a wild card is used. A wild card represents any concept in the given substrate. In the brief syntax, a wildcard is represented using the "*" symbol. In the full syntax, the text "ANY" (case insensitive) is also allowed.

focusConcept = [memberOf ws] (conceptReference / wildCard)

$$crOrWildCard ::= cr\langle\!\langle conceptReference \rangle\!\rangle \mid wc$$
$$focusConcept ::=$$
$$focusConcept\_m\langle\!\langle crOrWildCard \rangle\!\rangle \mid$$
$$focusConcept\_c\langle\!\langle crOrWildCard \rangle\!\rangle$$

Interpretation: If memberOf is present the interpretation of focusConcept is union the interpretation of memberOf applied to each element in the interpretation of conceptReference. Note that the interpretation of wildCard is different in the context of memberOf than it is outside. memberOf wildCard is interpreted as the interpretation of the domain of the *refset* function (i.e. all of the refsets in the substrate), where, without, it is interpreted as *all* of the concepts in the system.

If memberOf isn't specified, the interpretation is the substrate interpretations of conceptReference itself

$i\_focusConcept : Substrate \rightarrow focusConcept \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ fc : focusConcept \bullet$
$i\_focusConcept\ ss\ fc =$
if $focusConcept\_c^{\sim}fc \in crOrWildCard$ **then**
    if $cr^{\sim}(focusConcept\_c^{\sim}fc) \in conceptReference$ **then**
        $ss.i\_conceptReference\ (cr^{\sim}(focusConcept\_c^{\sim}fc))$
    else
        $ok\ ss.concepts$
else
    $i\_memberOf\ ss\ (focusConcept\_m^{\sim}fc)$

4.6.1   memberOf

MEMBEROF   returns the union of the application of the substrate *refset* function
to a wild card or the the supplied reference set identifiers. An error is returned
if (a) *refsetids* already has an error or (b) the substrate interpretation of a given
refsetId returns an error.

$i\_memberOf : Substrate \rightarrow crOrWildCard \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ crorwc : crOrWildCard \bullet$
    $i\_memberOf\ ss\ crorwc =$
$(\textbf{let}\ refsetids ==$
    if $cr^{\sim}crorwc \in conceptReference$
    then
        $ss.i\_conceptReference(cr^{\sim}crorwc)$
    else
        $ok\ (ss.descendants\ refset\_concept) \bullet$
    if $refsetids \in \operatorname{ran} error$
    then
        $refsetids$
    else
        $bigunion\{sctid : result\_sctids\ refsetids \bullet ss.i\_refsetId\ sctid\})$

5 Types

This section carries various type transformations and error checking functions

5.1 Types

5.1.1 Quads_or_Error

Quads_or_Error is a collection of *Quad*s or an error condition. If it is a collection of *Quad*s, it also carries a direction indicator that determines whether it is the source or targets that carry the matching elements. Note:

- *source_direction* – matches were performed on attribute/target. Return the source *sctId*s.
- *targets_direction* – matches were performed on attribute/source (REVERSEFLAG was present). Return the target *sctId*s or, eventually, concrete values

$$direction ::= source\_direction \mid targets\_direction$$
$$Quads\_or\_Error ::= quad\_value\langle\!\langle \mathbb{P}\ Quad \times direction \rangle\!\rangle \mid qerror\langle\!\langle ERROR \rangle\!\rangle$$

5.1.2 sctIdGroups

The *groupId* identifies a set of one or more quads associated with the same subject. The zero group, however, is treated differently, where each individual quad is treated as a group unto itself. To implement this, we have to assign some sort of reproducible secondary "group" identifier to zero groups. While actual implementations will probably use a secondary identifier such as the RF2 relationship sctid, for the purposes of this specification we use the quad itself as the unique identity of the "group". This is represented by the *uniqueGroupId* construct below:

$$uniqueGroupId ::= ug\langle\!\langle groupId \rangle\!\rangle \mid zg\langle\!\langle Quad \rangle\!\rangle$$

The *sctidGroup* represents a subject or target *sctid*s that meet the requirements of an attribute or attribute set within the context of a group. Each *sctidGroup* entry carries with the *sctid* that passed along with the group identifier of the group in which it passed, or the whole Quad in the case of zero groups.of the specific group in which it passed. The *sctidGroup* construct is used for unions, intersections and differences within the context of a specific group.

sctIdGroups_or_Error represents a set of *sctIdGroups* or an error condition

$$sctIdGroup == sctId \times uniqueGroupId$$
$$sctIdGroups\_or\_Error ::= group\_value\langle\!\langle \mathbb{P}\ sctIdGroup \rangle\!\rangle \mid$$
$$gerror\langle\!\langle ERROR \rangle\!\rangle$$

The *quadGroup* function converts a *Quad* to a corresponding *uniqueGroupId*, where the result is the *groupId* of the quad if the group is non-zero, and the quad itself when the group is zero.

$$\begin{array}{|l}
quadGroup : Quad \rightarrow uniqueGroupId \\
\hline
\forall\, q : Quad \bullet quadGroup\ q = \\
\quad \textbf{if}\ q.g \neq zero\_group\ \textbf{then} \\
\qquad ug\ q.g \\
\quad \textbf{else} \\
\qquad zg\ q
\end{array}$$

5.2 Result transformations

- **result_sctids** – the set of *sctId*s in *Sctids_or_Error* or the empty set if there is an error
- **quads_for** – the set of quads in a *Quads_or_Error* or an empty set if there is an error
- **quad_direction** – the direction of a *Quads_or_Error* result. Undefined if error

$$\begin{array}{|l}
result\_sctids : Sctids\_or\_Error \rightarrow \mathbb{P}\, sctId \\
quads\_for : Quads\_or\_Error \rightarrow \mathbb{P}\, Quad \\
quad\_direction : Quads\_or\_Error \nrightarrow direction \\
\hline
\forall\, r : Sctids\_or\_Error \bullet result\_sctids\ r = \\
\quad \textbf{if}\ r \in \mathrm{ran}\ error\ \textbf{then}\ \emptyset \\
\quad \textbf{else}\ ok^{\sim} r \\
\\
\forall\, q : Quads\_or\_Error \bullet quads\_for\ q = \\
\quad \textbf{if}\ q \in \mathrm{ran}\ qerror\ \textbf{then}\ \emptyset \\
\quad \textbf{else}\ first\,(quad\_value^{\sim} q) \\
\\
\forall\, q : Quads\_or\_Error \bullet quad\_direction\ q = \\
\quad second\,(quad\_value^{\sim} q)
\end{array}$$

Definition of the various functions that are performed on the result type.

- **firstError** – aggregate one or more *Sctids_or_Error* types, at least one of which carries and error and merge them into a single *Sctid_or_Error* instance propagating at least one of the errors (Not fully defined)
- **gfirstError** – convert two *sctIdGroups_or_Error* into a single *sctIdGroups_or_Error* propagating at least one of the errors.

- **union** – return the union of two *Sctids_or_Error* types, propagating errors if they exist, else returning the union of the sctId sets.
- **intersect** –return the intersection of two *Sctids_or_Error* types, propagating errors if they exist, else returning the intersection of the sctId sets.
- **minus** – return the difference of one *Sctids_or_Error* type and a second, propagating errors if they exist, else returning the set of sctId's in the first set that aren't in the second.

- **bigunion** – return the union of a set of *Sctids_or_Error* types, propagating errors if they exist, else returning the union of all of the sctId sets.
- **bigintersect** – return the intersection a set of *Sctids_or_Error* types, propagating errors if they exist, else returning the intersection of all of the sctId sets.

$firstError : \mathbb{P}\,Sctids\_or\_Error \rightarrow Sctids\_or\_Error$
$gfirstError : \mathbb{P}\,sctIdGroups\_or\_Error \rightarrow sctIdGroups\_or\_Error$

$union, intersect, minus : Sctids\_or\_Error \rightarrow Sctids\_or\_Error \rightarrow$
$\qquad Sctids\_or\_Error$
$bigunion, bigintersect : \mathbb{P}\,Sctids\_or\_Error \rightarrow Sctids\_or\_Error$

$gunion, gintersect, gminus :$
$\qquad sctIdGroups\_or\_Error \rightarrow sctIdGroups\_or\_Error \rightarrow sctIdGroups\_or\_Error$

$\forall\, x, y : Sctids\_or\_Error \bullet union\ x\ y =$
$\qquad \textbf{if } x \in \mathrm{ran}\,error \vee y \in \mathrm{ran}\,error \textbf{ then } firstError\ \{x, y\}$
$\qquad \textbf{else } ok\,((ok^\sim x) \cup (ok^\sim y))$

$\forall\, x, y : Sctids\_or\_Error \bullet intersect\ x\ y =$
$\qquad \textbf{if } x \in \mathrm{ran}\,error \vee y \in \mathrm{ran}\,error \textbf{ then } firstError\ \{x, y\}$
$\qquad \textbf{else } ok\,((ok^\sim x) \cap (ok^\sim y))$

$\forall\, x, y : Sctids\_or\_Error \bullet minus\ x\ y =$
$\qquad \textbf{if } x \in \mathrm{ran}\,error \vee y \in \mathrm{ran}\,error \textbf{ then } firstError\ \{x, y\}$
$\qquad \textbf{else } ok\,((ok^\sim x) \setminus (ok^\sim y))$

$\forall\, rs : \mathbb{P}\,Sctids\_or\_Error \bullet bigunion\ rs =$
$\qquad \textbf{if } \exists\, r : rs \bullet r \in \mathrm{ran}\,error \textbf{ then } firstError\ rs$
$\qquad \textbf{else } ok\,(\bigcup\{r : rs \bullet result\_sctids\ r\})$

$\forall\, rs : \mathbb{P}\,Sctids\_or\_Error \bullet bigintersect\ rs =$
$\qquad \textbf{if } \exists\, r : rs \bullet r \in \mathrm{ran}\,error \textbf{ then } firstError\ rs$
$\qquad \textbf{else } ok\,(\bigcap\{r : rs \bullet result\_sctids\ r\})$

$\forall\, x, y : sctIdGroups\_or\_Error \bullet gintersect\ x\ y =$
$\qquad \textbf{if } x \in \mathrm{ran}\,gerror \vee y \in \mathrm{ran}\,gerror \textbf{ then } gfirstError\ \{x, y\}$
$\qquad \textbf{else } group\_value((group\_value^\sim x) \cap (group\_value^\sim y))$

$\forall\, x, y : sctIdGroups\_or\_Error \bullet gunion\ x\ y =$
$\qquad \textbf{if } x \in \mathrm{ran}\,gerror \vee y \in \mathrm{ran}\,gerror \textbf{ then } gfirstError\ \{x, y\}$
$\qquad \textbf{else } group\_value((group\_value^\sim x) \cup (group\_value^\sim y))$

$\forall\, x, y : sctIdGroups\_or\_Error \bullet gminus\ x\ y =$
$\qquad \textbf{if } x \in \mathrm{ran}\,gerror \vee y \in \mathrm{ran}\,gerror \textbf{ then } gfirstError\ \{x, y\}$
$\qquad \textbf{else } group\_value((group\_value^\sim x) \setminus (group\_value^\sim y))$

A Optional elements

Representing optional elements of type T. Representing it as a sequence allows us to determine absence by $\#T = 0$ and the value by *head T*.

$$T[0 \mathbin{.\,.} 1] == \{s : \text{seq } T \mid \#s \leq 1\}$$

B Generic cardinality evaluation

evalCardinality Evaluate the cardinality of an arbitrary set of type T. If the number of elements in the set of T is greater than *min* and:

- *max* is a number and the number of elements in T is less than *max*
- *max* is *many*

Then return the the set. Otherwise return the empty set.

$$
\begin{array}{l}
[T] \\
\hline
evalCardinality : \mathbb{N} \to unlimitedNat \to \mathbb{P}\, T \to \mathbb{P}\, T \\
\hline
\forall\, min : \mathbb{N};\ max : unlimitedNat;\ t : \mathbb{P}\, T \bullet \\
evalCardinality\ min\ max\ t = \\
\quad \textbf{if}\ (\#t \geq min) \wedge (max = many \vee \#t \leq (num\tilde{}\,max)) \\
\quad \textbf{then} \\
\qquad t \\
\quad \textbf{else} \\
\qquad \emptyset
\end{array}
$$

C Generic sequence function

A generic function that takes:

- A substrate
- A function that takes a substrate, a sequence of type T and returns *Sctids_or_Error* (example: *i_subExpressionConstraint*)
- An operator that takes two *Sctids_or_Error* and returns a combination (example: *union*)
- A tuple of the form "$T \times \text{seq}_1\ T$"

And returns *Sctids_or_Error*

In the formalization below, *first seq_e* refers to the left hand side of the $T \times \text{seq}_1\ T$ and *second seq_e* to the right hand side. *head(second seq_e)* refers to the first element in the sequence and *tail(second seq_e)* refers to the remaining elements in the sequence, which may be empty ($\langle\rangle$).

$[T]$

$applyToSequence : Substrate \rightarrow (Substrate \rightarrow T \rightarrow Sctids\_or\_Error) \rightarrow$
$\quad (Sctids\_or\_Error \rightarrow Sctids\_or\_Error \rightarrow Sctids\_or\_Error) \rightarrow$
$\quad (T \times \mathrm{seq}_1\ T) \rightarrow Sctids\_or\_Error$

$\forall\, ss : Substrate;\ f : (Substrate \rightarrow T \rightarrow Sctids\_or\_Error);$
$\quad op : (Sctids\_or\_Error \rightarrow Sctids\_or\_Error \rightarrow Sctids\_or\_Error);$
$\quad seq\_e : (T \times \mathrm{seq}_1\ T) \bullet$
$applyToSequence\ ss\ f\ op\ seq\_e =$
$\mathbf{if}\ tail(second\ seq\_e) = \langle\rangle\ \mathbf{then}$
$\quad op\ (f\ ss\ (first\ seq\_e))(f\ ss\ (head\ (second\ seq\_e)))$
$\mathbf{else}$
$\quad op\ (f\ ss\ (first\ seq\_e))(applyToSequence\ ss\ f\ op\ (head\ (second\ seq\_e), tail\ (second\ seq\_e)))$