

A Declarative Semantics for SNOMED CT Expression Constraints

March 16, 2015

Contents

1	Axiomatic Data Types	2
1.1	Atomic Data Types	2
1.2	Composite Data Types	3
2	The Substrate	3
2.1	Substrate Components	3
2.1.1	conceptReference	3
2.2	Substrate	4
2.2.1	Strict and Permissive Substrates	7
2.2.2	Strict Substrate	7
2.2.3	Permissive Substrate	7
3	Interpretation of Expression Constraints	7
3.1	Interpretation Output	8
3.2	expressionConstraint	8
3.2.1	unrefinedExpressionConstraint	8
3.2.2	refinedExpressionConstraint	9
3.2.3	simpleExpressionConstraint	10
3.2.4	compoundExpressionConstraint	10
3.2.5	conjunctionExpressionConstraint	11
3.2.6	disjunctionExpressionConstraint	11
3.2.7	exclusionExpressionConstraint	12
3.2.8	subExpressionConstraint	12
3.3	refinement	13
3.3.1	conjunctionRefinementSet	13
3.3.2	disjunctionRefinementSet	14
3.3.3	subRefinement	14
3.4	attributeSet	15
3.4.1	conjunctionAttributeSet	16
3.4.2	disjunctionAttributeSet	16
3.4.3	subAttributeSet	16

3.5	attributeGroup	17
3.5.1	attributeSet for Groups	17
3.5.2	conjunctionAttributeSet for Groups	18
3.5.3	disjunctionAttributeSet for Groups	18
3.5.4	subAttributeSet for Groups	18
3.6	attribute	18
3.6.1	Attribute Cardinality Interpretation	19
3.7	Cardinality	19
3.7.1	expressionAttribute	20
3.7.2	concreteAttribute	21
3.8	Group Cardinality	21
4	Substrate Interpretations	22
4.1	attributeExpressionConstraint	23
4.2	concreteAttributeConstraint	23
4.3	FocusConcept	24
4.3.1	focusConcept	25
4.3.2	memberOf	25
5	Glue and Helper Functions	25
5.1	Types	25
5.1.1	Quads_or_Error	25
5.1.2	IDGroups	26
5.2	Result transformations	26
6	Appendix 1 – Optional elements	29
7	Appendix 2 – Generic cardinality evaluation	30
8	Appendix 3 - Generic sequence function	30

1 Axiomatic Data Types

1.1 Atomic Data Types

This section identifies the atomic data types that are assumed for the rest of this specification, specifically:

- **sctId** – a SNOMED CT identifier
- **term** – a fully specified name, preferred term or synonym for a SNOMED CT Concept
- **decimalValue** – a decimal number
- **stringValue** – a string literal
- **groupId** – a role group identifier
- \mathbb{N} – a non-negative integer (built in to Z)
- \mathbb{Z} – an integer (built in to Z)

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

We will also need to recognize 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

While we can't fully specify the behavior of the concrete data types portion of the specification at this point, it is still useful to spell out the anticipated behavior on an abstract level.

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

$concreteValue ::=$
 $cv_string\langle\langle stringValue \rangle\rangle \mid cv_integer\langle\langle \mathbb{Z} \rangle\rangle \mid cv_decimal\langle\langle decimalValue \rangle\rangle$
 $target ::= t_sctid\langle\langle sctId \rangle\rangle \mid t_concrete\langle\langle concreteValue \rangle\rangle$

2 The Substrate

A substrate represents the context of an interpretation.

2.1 Substrate Components

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*)

$Quad$ $s : sctId$ $a : sctId$ $t : target$ $g : groupId$	$a = is_a \Rightarrow (g = zero_group \wedge t \in \text{ran } t_sctid)$
-----------------------------------------------------------------------	-----------------------------------------------------------------------------

2.1.1 conceptReference

The root of the expression constraint language is concept references – textual representations SNOMED CT identifiers accompanied by an optional *term* that

conveys their intended meaning to the human reader. **term** is ignored for the purposes of interpretation.

```
conceptReference = conceptId [ws "—" ws term ws "—"]
conceptId = sctId
```

```
conceptId == sctId
conceptReference == conceptId × term[0..1]
attributeName == conceptReference
```

2.2 Substrate

A substrate consists of:

- **concepts** The set of *sctIds* (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 (*c1* 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

$concepts : \mathbb{P} \text{ sctId}$
 $relationships : \mathbb{P} \text{ Quad}$
 $parentsOf : \text{ sctId} \rightarrow \mathbb{P} \text{ sctId}$
 $equivalent_concepts : \text{ sctId} \rightarrow \mathbb{P} \text{ sctId}$
 $refsets : \text{ sctId} \rightarrow \mathbb{P} \text{ sctId}$

 $childrenOf : \text{ sctId} \rightarrow \mathbb{P} \text{ sctId}$
 $descendants : \text{ sctId} \rightarrow \mathbb{P} \text{ sctId}$
 $ancestors : \text{ sctId} \rightarrow \mathbb{P} \text{ sctId}$

 $i_conceptReference : \text{ conceptReference} \rightarrow \text{ Sctids_or_Error}$
 $i_attributeName : \text{ attributeName} \rightarrow \text{ Sctids_or_Error}$
 $i_refsetId : \text{ sctId} \rightarrow \text{ Sctids_or_Error}$

$\forall rel : relationships \bullet rel.s \in concepts \wedge rel.a \in concepts \wedge$
 $(rel.t \in \text{ran } t_sctid \Rightarrow t_sctid \sim rel.t \in concepts)$

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

 $\forall r : relationships \bullet r.a = is_a \Rightarrow (t_sctid \sim r.t) \in parentsOf \ r.s$

 $\text{dom } equivalent_concepts = concepts$
 $\bigcup(\text{ran } equivalent_concepts) \subseteq concepts$
 $\forall c : concepts \bullet c \in equivalent_concepts \ c$
 $\forall c1, c2 : concepts \mid c2 \in (equivalent_concepts \ c1) \bullet$
 $\quad parentsOf \ c1 = parentsOf \ c2 \wedge$
 $\quad \{r : relationships \mid r.s = c1\} = \{r : relationships \mid r.s = c2\} \wedge$
 $\quad \{r : relationships \mid r.a = c1\} = \{r : relationships \mid r.a = c2\} \wedge$
 $\quad \{r : relationships \mid t_sctid \sim r.t = c1\} = \{r : relationships \mid t_sctid \sim r.t = c2\} \wedge$
 $\quad c1 \in \text{dom } refsets \Leftrightarrow c2 \in \text{dom } refsets \wedge$
 $\quad c1 \in \text{dom } refsets \Rightarrow refsets \ c1 = refsets \ c2 \wedge$
 $\quad (\forall rsd : \text{ran } refsets \bullet c1 \in rsd \Leftrightarrow c2 \in rsd)$

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

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

 $\forall s : concepts \bullet$
 $\quad descendants \ s = childrenOf \ s \cup \bigcup\{t : childrenOf \ s \bullet descendants \ t\}$
 $\forall t : concepts \bullet$
 $\quad ancestors \ t = parentsOf \ t \cup \bigcup\{s : parentsOf \ t \bullet ancestors \ s\}$
 $\forall t : concepts \bullet t \notin ancestors \ t$

 $\forall cr_interp : \text{ran } i_conceptReference \mid cr_interp \in \text{ran } ok \bullet$
 $\quad result_sctids \ cr_interp \subseteq concepts$
 $\forall att_interp : \text{ran } i_attributeName \mid att_interp \in \text{ran } ok \bullet$
 $\quad result_sctids \ att_interp \subseteq concepts$
 $\forall refset_interp : \text{ran } i_refsetId \mid refset_interp \in \text{ran } ok \bullet$
 $\quad result_sctids \ refset_interp \subseteq concepts$

2.2.1 Strict and Permissive Substrates

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 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 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

<i>strict_substrate</i>
<div style="border-bottom: 1px solid black; margin-bottom: 10px;"> <i>Substrate</i> </div> <pre> ∀ cr : conceptReference • i_conceptReference cr = if first cr ∉ concepts then error unknownConceptReference else ok (equivalent_concepts (first cr)) ∀ an : attributeName • i_attributeName an = (let rslt == i_conceptReference an • if rslt ∈ ran error then rslt else if result_sctids rslt ⊆ (descendants attribute_concept) then rslt else error unknownAttributeId) ∀ rsid : sctId • i_refsetId rsid = if rsid ∈ descendants refset_concept then ok {rsid} else error unknownRefsetId </pre>

2.2.3 Permissive Substrate

(fill in options)

3 Interpretation of Expression Constraints

An **expressionConstraint** is interpreted in the context of a *Substrate* and returns a set of *sctIds* 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*

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

$$Sctids_or_Error ::= ok\langle\langle \mathbb{P} \text{ sctId} \rangle\rangle \mid error\langle\langle ERROR \rangle\rangle$$

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

```
expressionConstraint = ws ( refinedExpressionConstraint / unrefinedExpressionConstraint )
ws
```

`expressionConstraint` takes either a `refinedExpressionConstraint` or `unrefinedExpressionConstraint` and returns its interpretation as either a set of `sctIds` or an error condition.

$$expressionConstraint ::= \\ expcons_refined\langle\langle refinedExpressionConstraint \rangle\rangle \mid \\ expcons_unrefined\langle\langle unrefinedExpressionConstraint \rangle\rangle$$

$$i_expressionConstraint : \\ Substrate \rightarrow expressionConstraint \rightarrow Sctids_or_Error \\ \\ \forall ss : Substrate; ec : expressionConstraint \bullet i_expressionConstraint ss ec = \\ \text{if } ec \in \text{ran } expcons_refined \\ \quad \text{then } i_refinedExpressionConstraint ss (expcons_refined \sim ec) \\ \quad \text{else } i_unrefinedExpressionConstraint ss (expcons_unrefined \sim ec)$$

3.2.1 unrefinedExpressionConstraint

The interpretation of an `unrefinedExpressionConstraint` is either the interpretation of a `compoundExpressionConstraint` or a `simpleExpressionConstraint`

$\text{unrefinedExpressionConstraint} = \text{compoundExpressionConstraint} / \text{simpleExpressionConstraint}$

$$\begin{aligned} \text{unrefinedExpressionConstraint} ::= & \\ & \text{unrefined_compound} \langle \langle \text{compoundExpressionConstraint} \rangle \rangle \mid \\ & \text{unrefined_simple} \langle \langle \text{simpleExpressionConstraint} \rangle \rangle \end{aligned}$$

$\begin{aligned} & i_unrefinedExpressionConstraint : \\ & \quad \text{Substrate} \rightarrow \text{unrefinedExpressionConstraint} \rightarrow \text{Sctids_or_Error} \end{aligned}$
$\begin{aligned} & \forall ss : \text{Substrate}; uec : \text{unrefinedExpressionConstraint} \bullet \\ & i_unrefinedExpressionConstraint ss uec = \\ & \text{if } uec \in \text{ran } \text{unrefined_compound} \\ & \quad \text{then } i_compoundExpressionConstraint ss (\text{unrefined_compound} \sim uec) \\ & \quad \text{else } i_simpleExpressionConstraint ss (\text{unrefined_simple} \sim uec) \end{aligned}$

3.2.2 refinedExpressionConstraint

$\begin{aligned} \text{refinedExpressionConstraint} = & \text{unrefinedExpressionConstraint ws ":" ws refinement} / \text{"(" ws} \\ & \text{refinedExpressionConstraint ws ")"} \end{aligned}$

The interpretation of **refinedExpressionConstraint** is the intersection 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 impact on the results.

$$\begin{aligned} \text{refinedExpressionConstraint} == & \\ & \text{unrefinedExpressionConstraint} \times \text{refinement} \\ & [\text{refinedExpressionConstraint}'] \end{aligned}$$

$\begin{aligned} & i_refinedExpressionConstraint : \\ & \quad \text{Substrate} \rightarrow \text{refinedExpressionConstraint} \rightarrow \text{Sctids_or_Error} \end{aligned}$
$\begin{aligned} & \forall ss : \text{Substrate}; rec : \text{refinedExpressionConstraint} \bullet \\ & i_refinedExpressionConstraint ss rec = \\ & \quad \text{intersect} (i_unrefinedExpressionConstraint ss (\text{first } rec)) (i_refinement ss (\text{second } rec)) \end{aligned}$

$\begin{aligned} & i_refinedExpressionConstraint' : \\ & \quad \text{Substrate} \rightarrow \text{refinedExpressionConstraint}' \rightarrow \text{Sctids_or_Error} \end{aligned}$

3.2.3 simpleExpressionConstraint

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 ws] focusConcept
bnfconstraintOperator = descendantOrSelfOf / descendantOf / ancestorOrSelfOf / ancestorOf
```

simpleExpressionConstraint == *constraintOperator*[0..1] × *focusConcept*
constraintOperator ::= *descendantOrSelfOf* | *descendantOf* | *ancestorOrSelfOf* | *ancestorOf*

i_simpleExpressionConstraint :
Substrate → *simpleExpressionConstraint* → *Sctids_or_Error*

∀ *ss* : *Substrate*; *sec* : *simpleExpressionConstraint* •
i_simpleExpressionConstraint ss sec =
i_constraintOperator ss (first sec) (i_focusConcept ss (second sec))

3.2.4 compoundExpressionConstraint

The interpretation of a *compoundExpressionConstraint* is the interpretation of its corresponding component.

```
compoundExpressionConstraint = F1mU7 ExpressionConstraint / disjunctionExpressionConstraint / exclusionExpressionConstraint / "(" ws compoundExpressionConstraint ws ")"
```

compoundExpressionConstraint ::=
compound_conj⟨⟨*conjunctionExpressionConstraint*⟩⟩ |
compound_disj⟨⟨*disjunctionExpressionConstraint*⟩⟩ |
compound_excl⟨⟨*exclusionExpressionConstraint*⟩⟩

i_compoundExpressionConstraint :
Substrate → *compoundExpressionConstraint* → *Sctids_or_Error*

∀ *ss* : *Substrate*; *cec* : *compoundExpressionConstraint* •
i_compoundExpressionConstraint ss cec =
if *cec* ∈ ran *compound_conj*
 then *i_conjunctionExpressionConstraint ss (compound_conj~cec)*
else if *cec* ∈ ran *compound_disj*
 then *i_disjunctionExpressionConstraint ss (compound_disj~cec)*
else *i_exclusionExpressionConstraint ss (compound_excl~cec)*

The signature below is used because the definition of `compoundExpressionConstraint` is recursive

$$\begin{array}{l} i_compoundExpressionConstraint' : \\ \quad Substrate \rightarrow compoundExpressionConstraint \rightarrow Sctids_or_Error \end{array}$$

3.2.5 conjunctionExpressionConstraint

`conjunctionExpressionConstraint` is interpreted the conjunction (intersection) of the interpretation of two or more `subExpressionConstraints`. The `conjunction` aspect is ignored because there is no other choice

$$conjunctionExpressionConstraint = subExpressionConstraint \ 1*(ws \ conjunction \ ws \ subExpressionConstraint)$$

$$conjunctionExpressionConstraint == \\ subExpressionConstraint \times seq_1(subExpressionConstraint)$$

Apply the intersection operator to the interpretation of each `subExpressionConstraint`

$$\begin{array}{l} i_conjunctionExpressionConstraint : \\ \quad Substrate \rightarrow conjunctionExpressionConstraint \rightarrow Sctids_or_Error \\ \hline \forall ss : Substrate; cecr : conjunctionExpressionConstraint \bullet \\ i_conjunctionExpressionConstraint \ ss \ cecr = \\ \quad applyToSequence \ ss \ i_subExpressionConstraint \ intersect \ cecr \end{array}$$

3.2.6 disjunctionExpressionConstraint

`disjunctionExpressionConstraint` is interpreted the disjunction (union) of the interpretation of two or more `subExpressionConstraints`. The `disjunction` element is ignored because there is no other choice.

$$disjunctionExpressionConstraint = subExpressionConstraint \ 1*(ws \ disjunction \ ws \ subExpressionConstraint)$$

$$disjunctionExpressionConstraint == \\ subExpressionConstraint \times seq_1(subExpressionConstraint)$$

Apply the union operator to the interpretation of each `subExpressionConstraint`

$i_disjunctionExpressionConstraint :$ $Substrate \rightarrow disjunctionExpressionConstraint \rightarrow Sctids_or_Error$
$\forall ss : Substrate; decr : disjunctionExpressionConstraint \bullet$ $i_disjunctionExpressionConstraint ss decr =$ $applyToSequence ss i_subExpressionConstraint union decr$

3.2.7 exclusionExpressionConstraint

The interpretation **exclusionExpressionConstraint** removes the interpretation of the second **exclusionExpressionConstraint** from the interpretation of the first. Errors are propagated.

$exclusionExpressionConstraint = subExpressionConstraint \text{ ws } exclusion \text{ ws } subExpressionConstraint$

$$exclusionExpressionConstraint ==$$

$$subExpressionConstraint \times subExpressionConstraint$$

$i_exclusionExpressionConstraint :$ $Substrate \rightarrow exclusionExpressionConstraint \rightarrow Sctids_or_Error$
$\forall ss : Substrate; ecr : exclusionExpressionConstraint \bullet$ $i_exclusionExpressionConstraint ss ecr =$ $minus (i_subExpressionConstraint ss (first ecr))(i_subExpressionConstraint ss (second ecr))$

3.2.8 subExpressionConstraint

subExpressionConstraint is interpreted as the interpretation of either a *simpleExpressionConstraint* or a *compoundExpressionConstraint*

$subExpressionConstraint = simpleExpressionConstraint / "(" \text{ ws } (compoundExpressionConstraint / refinedExpressionConstraint) \text{ ws } ")"$

$$subExpressionConstraint ::=$$

$$subExpr_simple \langle \langle simpleExpressionConstraint \rangle \rangle \mid$$

$$subExpr_compound \langle \langle compoundExpressionConstraint \rangle \rangle \mid$$

$$subExpr_refined \langle \langle refinedExpressionConstraint' \rangle \rangle$$

$i_subExpressionConstraint :$ $Substrate \rightarrow subExpressionConstraint \rightarrow Sctids_or_Error$
$\forall ss : Substrate; sec : subExpressionConstraint \bullet$ $i_subExpressionConstraint ss sec =$ if $sec \in \text{ran } subExpr_simple$ then $i_simpleExpressionConstraint ss (subExpr_simple \sim sec)$ else if $sec \in \text{ran } subExpr_compound$ then $i_compoundExpressionConstraint' ss (subExpr_compound \sim sec)$ else $i_refinedExpressionConstraint' ss (subExpr_refined \sim sec)$

3.3 refinement

The interpretation of **refinement** is the interpretation of the **subRefinement**, **conjunctionGroup** or **disjunctionGroup**

$\text{refinement} = \text{subRefinement} [\text{conjunctionRefinementSet} / \text{disjunctionRefinementSet}]$

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

$\text{refinementConjunctionOrDisjunction} ::=$
 $\text{refine_conjset} \langle \langle \text{conjunctionRefinementSet} \rangle \rangle \mid$
 $\text{refine_disjset} \langle \langle \text{disjunctionRefinementSet} \rangle \rangle$

$i_refinement : Substrate \rightarrow refinement \rightarrow Sctids_or_Error$
$\forall ss : Substrate; rfnment : refinement \bullet$ $i_refinement ss rfnment =$ $(\text{let } lhs == i_subRefinement ss (\text{first } rfnment); rhs == \text{second } rfnment \bullet$ if $\#rhs = 0$ then lhs else if $(\text{head } rhs) \in \text{ran } \text{refine_conjset}$ then $\text{intersect } lhs (i_conjunctionRefinementSet ss (\text{refine_conjset} \sim (\text{head } rhs)))$ else $\text{union } lhs (i_disjunctionRefinementSet ss (\text{refine_disjset} \sim (\text{head } rhs)))$

$i_refinement' : Substrate \rightarrow refinement' \rightarrow Sctids_or_Error$

3.3.1 conjunctionRefinementSet

$$\text{conjunctionRefinementSet} = 1^*(\text{ws conjunction ws subRefinement})$$

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

Apply the intersect operator to the interpretation of each subRefinement

i_conjunctionRefinementSet :

Substrate \rightarrow *conjunctionRefinementSet* \rightarrow *Sctids_or_Error*

$\forall ss : \text{Substrate}; \text{conjset} : \text{conjunctionRefinementSet} \bullet$
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

$$\text{disjunctionRefinementSet} = 1^*(\text{ws disjunction ws subRefinement})$$

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

Apply the union operator to the interpretation of each subRefinement

i_disjunctionRefinementSet :

Substrate \rightarrow *disjunctionRefinementSet* \rightarrow *Sctids_or_Error*

$\forall ss : \text{Substrate}; \text{disjset} : \text{disjunctionRefinementSet} \bullet$
i_disjunctionRefinementSet *ss* *disjset* =
if *tail disjset* = $\langle \rangle$
 then *i_subRefinement* *ss* (*head disjset*)
else
 intersect (*i_subRefinement* *ss* (*head disjset*)) (*i_disjunctionRefinementSet* *ss* (*tail disjset*))

3.3.3 subRefinement

The interpretation of a **subRefinement** is the interpretation of the corresponding **attributeSet**, **attributeGroup** or **refinement**.

$$\text{subRefinement} = \text{attributeSet} / \text{attributeGroup} / "(" \text{ ws refinement ws } ")"$$

```

subRefinement ::=
  subrefine_attset⟨⟨attributeSet⟩⟩ |
  subrefine_attgroup⟨⟨attributeGroup⟩⟩ |
  subrefine_refinement⟨⟨refinement'⟩⟩

```

$i_subRefinement :$ $Substrate \rightarrow subRefinement \rightarrow Sctids_or_Error$
$\forall ss : Substrate; subrefine : subRefinement \bullet$ $i_subRefinement ss subrefine =$ if $subrefine \in \text{ran } subrefine_attset$ then $i_attributeSet ss (subrefine_attset \sim subrefine)$ else if $subrefine \in \text{ran } subrefine_attgroup$ then $i_attributeGroup ss (subrefine_attgroup \sim subrefine)$ else $i_refinement' ss (subrefine_refinement \sim subrefine)$

3.4 attributeSet

$attributeSet = subAttributeSet [\text{conjunctionAttributeSet} / \text{disjunctionAttributeSet}]$

```

attributeSet == subAttributeSet × conjunctionOrDisjunctionAttributeSet[0 .. 1]
[attributeSet']

```

```

conjunctionOrDisjunctionAttributeSet ::=
  attset_conjattset⟨⟨conjunctionAttributeSet⟩⟩ |
  attset_disjattset⟨⟨disjunctionAttributeSet⟩⟩

```

$i_attributeSet :$ $Substrate \rightarrow attributeSet \rightarrow Sctids_or_Error$
$\forall ss : Substrate; attset : attributeSet \bullet$ $i_attributeSet ss attset =$ (let $lhs == i_subAttributeSet ss (first attset); rhs == second attset \bullet$ if $\#rhs = 0$ then lhs else if $head rhs \in \text{ran } attset_conjattset$ then $intersect lhs (i_conjunctionAttributeSet ss (attset_conjattset \sim (head rhs)))$ else $union lhs (i_disjunctionAttributeSet ss (attset_disjattset \sim (head rhs)))$

$i_attributeSet' :$ $Substrate \rightarrow attributeSet' \rightarrow Sctids_or_Error$

3.4.1 conjunctionAttributeSet

$$\text{conjunctionAttributeSet} = 1^*(\text{ws conjunction ws subAttributeSet})$$

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

Apply the intersect operator to the interpretation of each subAttributeSet

$i_conjunctionAttributeSet :$
 $Substrate \rightarrow conjunctionAttributeSet \rightarrow Sctids_or_Error$

$\forall ss : Substrate; conjset : conjunctionAttributeSet \bullet$
 $i_conjunctionAttributeSet ss conjset =$
if $tail conjset = \langle \rangle$
 then $i_subAttributeSet ss (head conjset)$
 else
 $intersect (i_subAttributeSet ss (head conjset)) (i_conjunctionAttributeSet ss (tail conjset))$

3.4.2 disjunctionAttributeSet

$$\text{disjunctionAttributeSet} = 1^*(\text{ws disjunction ws subAttributeSet})$$

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

Apply the intersect operator to the interpretation of each subAttributeSet

$i_disjunctionAttributeSet :$
 $Substrate \rightarrow disjunctionAttributeSet \rightarrow Sctids_or_Error$

$\forall ss : Substrate; disjset : disjunctionAttributeSet \bullet$
 $i_disjunctionAttributeSet ss disjset =$
if $tail disjset = \langle \rangle$
 then $i_subAttributeSet ss (head disjset)$
 else
 $union (i_subAttributeSet ss (head disjset)) (i_disjunctionAttributeSet ss (tail disjset))$

3.4.3 subAttributeSet

$$\text{subAttributeSet} = \text{attribute} / \text{"(ws attributeSet ws)"}^*$$

$\text{subAttributeSet} ::=$
 $\text{subaset_attribute} \langle \langle \text{attribute} \rangle \rangle \mid$
 $\text{subaset_attset} \langle \langle \text{attributeSet}' \rangle \rangle$

$i_subAttributeSet :$ $Substrate \rightarrow subAttributeSet \rightarrow Sctids_or_Error$
$\forall ss : Substrate; subaset : subAttributeSet \bullet$ $i_subAttributeSet ss subaset =$ if $subaset \in \text{ran } subaset_attribute$ then $i_attribute ss (subaset_attribute \sim subaset)$ else $i_attributeSet' ss (subaset_attset \sim subaset)$

3.5 attributeGroup

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 (non-zero) group. 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 *IDGroups* rather than *Quads*

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

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

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

$i_attributeGroup : Substrate \rightarrow attributeGroup \rightarrow Sctids_or_Error$
$\forall ss : Substrate; ag : attributeGroup \bullet$ $i_attributeGroup ss ag =$ $idgroups_to_sctids (i_cardinality (first ag) (idgroups_to_sctids (i_groupAttributeSet ss (second ag)))$

3.5.1 attributeSet for Groups

The interpretation of an **attributeSet** within the context of a group is the interpretation of the **subAttributeSet** optionally intersected or union with additional conjunction or disjunction attribute sets.

$i_groupAttributeSet :$ $Substrate \rightarrow attributeSet \rightarrow IDGroups$
$\forall ss : Substrate; attset : attributeSet \bullet$ $i_groupAttributeSet ss attset =$ $(\text{let } lhs == i_groupSubAttributeSet ss (first attset); rhs == second attset \bullet$ $\text{if } \#rhs = 0$ $\text{then } lhs$ $\text{else if } head rhs \in \text{ran } attset_conjattset$ $\text{then } gintersect lhs (i_groupConjunctionAttributeSet ss (attset_conjattset \sim (head rhs)))$ $\text{else } gunion lhs (i_groupDisjunctionAttributeSet ss (attset_conjattset \sim (head rhs))))$

3.5.2 conjunctionAttributeSet for Groups

Apply the intersect operator to the interpretation of each subAttributeSet in a group context

$i_groupConjunctionAttributeSet :$ $Substrate \rightarrow conjunctionAttributeSet \rightarrow IDGroups$
$\forall ss : Substrate; conjset : conjunctionAttributeSet \bullet$ $i_groupConjunctionAttributeSet ss conjset =$ $\text{if } tail conjset = \langle \rangle$ $\text{then } i_groupSubAttributeSet ss (head conjset)$ else $gintersect (i_groupSubAttributeSet ss (head conjset)) (i_groupConjunctionAttributeSet ss (tail conjset))$

3.5.3 disjunctionAttributeSet for Groups

Apply the union operator to the interpretation of each subAttributeSet in a group context

$i_groupDisjunctionAttributeSet :$ $Substrate \rightarrow disjunctionAttributeSet \rightarrow IDGroups$
$\forall ss : Substrate; disjset : disjunctionAttributeSet \bullet$ $i_groupDisjunctionAttributeSet ss disjset =$ $\text{if } tail disjset = \langle \rangle$ $\text{then } i_groupSubAttributeSet ss (head disjset)$ else $gunion (i_groupSubAttributeSet ss (head disjset)) (i_groupDisjunctionAttributeSet ss (tail disjset))$

3.5.4 subAttributeSet for Groups

$i_groupSubAttributeSet :$ $Substrate \rightarrow subAttributeSet \rightarrow IDGroups$
$\forall ss : Substrate; subaset : subAttributeSet \bullet$ $i_subAttributeSet ss subaset =$ if $subaset \in \text{ran } subaset_attribute$ then $i_groupAttributess(subaset_attribute \sim subaset)$ else $i_groupAttributeSet' ss (subaset_attset \sim subaset)$

3.6 attribute

The interpretation of an attribute is the place where we transition from a set of *Quads* to a set of *sctIds*. Everything to the right hand side of **cardinality** is treated as quads. *i_cardinality* converts quads to *sctIds*

```
attribute = [cardinality ws] [reverseFlag ws] ws attributeName ws
(concreteComparisonOperator ws concreteValue /
expressionComparisonOperator ws expressionConstraintValue )
cardinality = "[" nonNegativeIntegerValue to (nonNegativeIntegerValue / many) "]"
```

$attribute$ $card : cardinality[0..1]$ $rf : reverseFlag[0..1]$ $name : attributeName$ $targets : concreteOrExpressionAttribute$

```
[reverseFlag]
concreteOrExpressionAttribute ::=
  attrib_conc⟨⟨concreteAttribute⟩⟩ |
  attrib_expr⟨⟨expressionAttribute⟩⟩

concreteAttribute == concreteComparisonOperator × concreteValue
expressionAttribute == expressionComparisonOperator × expressionConstraintValue
```

$i_attribute :$ $Substrate \rightarrow attribute \rightarrow Sctids_or_Error$
$\forall ss : Substrate; att : attribute \bullet$ $i_attribute\ ss\ att =$ $(let\ att_sctids == ss.i_attributeName\ att.name \bullet$ $if\ att_sctids \in ran\ error$ $\quad then\ att_sctids$ $else\ if\ att.targets \in ran\ attrib_conc$ $\quad then$ $\quad (let\ conc_interp ==$ $\quad i_concreteAttribute\ ss\ (result_sctids\ att_sctids)\ (attrib_conc \sim att.targets) \bullet$ $\quad i_cardinality\ att.card\ conc_interp)$ $else$ $\quad (let\ exp_interp ==$ $\quad i_expressionAttribute\ ss\ att.rf\ (result_sctids\ att_sctids)\ (attrib_expr \sim att.targets) \bullet$ $\quad i_cardinality\ att.card\ exp_interp))$

3.6.1 Attribute Cardinality Interpretation

For the sake of simplicity, we separate out the components of the concrete and expression constraints.

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

$$cardinality == \mathbb{N} \times unlimitedNat$$

3.7 Cardinality

This is the interpretation of non-grouped cardinality. It function that takes:

- An optional cardinality
- *Quads_or_Error*: which is one of:
 - A set of *Quads* and a direction indicator
 - An error

This function:

- Propagates an error condition if one exists
- Returns the empty set if the set of *Quads* falls outside the cardinality rules
- Returns the set of source *sctIds* in the *Quads* if the direction is forward (?s attribute t) (*source_direction*)
- Returns the set of target *sctIds* in the *Quads* if the direction is reverse (s attribute ?t) (*targets_direction*)

$i_cardinality :$ $cardinality[0 \dots 1] \rightarrow Quads_or_Error \rightarrow Sctids_or_Error$
$\forall card : cardinality[0 \dots 1]; gore : Quads_or_Error \bullet$ $i_cardinality\ card\ gore =$ if $gore \in \text{ran } qerror$ then $error(qerror \sim gore)$ else (let $qr == evalCardinality[Quad]\ card\ (quads_for\ gore) \bullet$ if $quad_direction\ gore = source_direction$ then $ok\ \{q : qr \bullet q.s\}$ else $ok\ \{q : qr \bullet t_sctid \sim q.t\}$)

3.7.1 expressionAttribute

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

expressionConstraintValue ::=
expression_simple $\langle\langle simpleExpressionConstraint \rangle\rangle$ |
expression_refined $\langle\langle refinedExpressionConstraint' \rangle\rangle$ |
expression_compound $\langle\langle compoundExpressionConstraint \rangle\rangle$

Expression attribute is a an additional expression we introduced to simplify interpretation. It is the interpretation of:

- An optional **reverseFlag**
- A set of attribute **sctIds**
- An **expressionAttribute** which consists of an **expressionComparisonOperator** and an **expressionConstraintValue**

The interpretation consists of the following steps:

1. Evaluate **expressionConstraintValue**
2. If the evaluation yielded an error propagate it
3. Evaluate the combination of the reverse flag, the attributes, the operator, the result of step 1 and return a set of quads with a direction indicator or an error

$i_expressionAttribute :$ $Substrate \rightarrow reverseFlag[0..1] \rightarrow \mathbb{P} \text{ sctId} \rightarrow expressionAttribute \rightarrow Quads_or_Error$
$\forall ss : Substrate; rf : reverseFlag[0..1]; atts : \mathbb{P} \text{ sctId}; ea : expressionAttribute \bullet$ $i_expressionAttribute ss rf atts ea =$ $(\text{let } target_sctids == \text{if } (second\ ea) \in \text{ran } expression_simple$ $\quad \text{then } i_simpleExpressionConstraint\ ss\ (expression_simple \sim (second\ ea))$ $\text{else if } (second\ ea) \in \text{ran } expression_refined$ $\quad \text{then } i_refinedExpressionConstraint'\ ss\ (expression_refined \sim (second\ ea))$ $\text{else } i_compoundExpressionConstraint\ ss\ (expression_compound \sim (second\ ea)) \bullet$ $i_attributeExpressionConstraint\ ss\ rf\ atts\ (first\ ea)\ target_sctids)$

3.7.2 concreteAttribute

```

concreteComparisonOperator = "=" / "!=" / "<>" / "<=" / "<" / ">=" / ">"
concreteValue = QM stringValue QM / "#" numericValue
stringValue = 1*(anyNonEscapedChar / escapedChar)
numericValue = decimalValue / integerValue

```

$concreteComparisonOperator ::=$
 $cco_eq \mid cco_neq \mid cco_leq \mid ccl_lt \mid cco_geq \mid cco_gt$

The interpretation of a **concreteAttribute** selects the set of quads in the substrate that have an attribute in the set of attributes determined by the interpretation of **attributeName** having *concreteValue* targets that meet the supplied comparison rules.

$i_concreteAttribute :$ $Substrate \rightarrow \mathbb{P} \text{ sctId} \rightarrow concreteAttribute \rightarrow Quads_or_Error$
$\forall ss : Substrate; attids : \mathbb{P} \text{ sctId}; ca : concreteAttribute \bullet$ $i_concreteAttribute ss attids ca =$ $i_concreteAttributeConstraint\ ss\ attids\ (first\ ca)\ (second\ ca)$

3.8 Group Cardinality

The interpretation of cardinality within a group impose additional constraints:

- $[0..n]$ – the set of all substrate concept codes that have at least one group (entry) in the substrate relationships and, at most n matching entries in the same group
- $[0..0]$ – the set of all substrate concept codes that have at least one group (entry) in the substrate relationships and *no* matching entries
- $[1..*]$ – (default) at least one matching entry in the substrate relationships

- $[m_1 \dots n_1]op[m_2 \dots n_2]\dots$ – set of substrate concept codes where there exists at least one group where all conditions are simultaneously true

The interpretation of a grouped cardinality is a function from a set of *sctId*'s to the groups in which they were qualified.

The algorithm below partitions the input set of *Quads* by group and validates the cardinality on a per-group basis. Groups that pass are returned

$i_groupCardinality :$ $cardinality[0 \dots 1] \rightarrow Quads_or_Error \rightarrow IDGroups$
$\forall oc : cardinality[0 \dots 1]; quads : Quads_or_error \bullet$ $i_groupCardinality\ oc\ quads =$ if $quads \in \text{ran } qerror$ then $qerror\ (qerror \sim quads)$ else $(\text{let } uniqueGroups == \{q : quads_for\ quads \mid q \neq zero_group \bullet q.g\};$ $passingGroups == \bigcup \{g : uniqueGroups \bullet evalCardinalityForGroup\ oc\ g\ (quads_for\ quads)\} \bullet group_va$

evalCardinalityForGroup takes an optional cardinality, a group id and collection of *Quads* and evaluates the cardinality against the subset of the *Quads* that belong to the supplied group.

| $noCardinality = \langle \rangle$

$evalCardinalityForGroup : cardinality[0 \dots 1] \rightarrow groupId \rightarrow \mathbb{P}\ Quad \rightarrow \mathbb{P}\ Quad$
$\forall oc : cardinality[0 \dots 1]; g : groupId; quads : \mathbb{P}\ Quad \bullet evalCardinalityForGroup\ oc\ g\ quads =$ $(\text{let } groupQuads = \{q : quads \mid q.g = g\} \bullet$ if $first\ oc = 0 \wedge (second\ oc \in \text{ran } num \wedge num \sim (second\ oc) = 0)$ then $\text{if } \#(evalCardinality[Quad]\ noCardinality\ groupQuads) > 0$ then $quads$ else \emptyset else if $first\ oc = 0$ then $evalCardinalityForGroup\ g\ quads = evalCardinality[Quad]\ oc\ \{q : quads \mid q.g = g\}$

quadsToTuples takes a collection of quads and returns a set of *sctId* \times *groupId* tuples.

$quadsToTuples : \mathbb{P}\ Quad \rightarrow direction \rightarrow \mathbb{P}(sctId \times groupId)$
$\forall quads : \mathbb{P}\ Quad; d : direction \bullet quadsToTuples\ quads\ d =$ if $direction = source_direction$ then $\{q : quads \bullet (q.s, q.g)\}$ else $\{q : quads \bullet (t_sctid\ q.t, q.g)\}$

4 Substrate Interpretations

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

4.1 attributeExpressionConstraint

```
expressionComparisonOperator = "=" / "!=" / "<>"
```

expressionComparisonOperator ::= *eco_eq* | *eco_neq*

attributeExpressionConstraint takes a substrate, an optional reverse flag, a set of attribute sctIds, an expression operator (equal or not equal) and a set of subject/target sctIds (depending on whether reverse flag is present) and returns a collection of quads that match / don't match the entry.

```
i_attributeExpressionConstraint :
  Substrate → reverseFlag[0..1] →  $\mathbb{P}$  sctId →
    expressionComparisonOperator → Sctids_or_Error → Quads_or_Error

 $\forall ss : \text{Substrate}; rf : \text{reverseFlag}[0..1]; atts : \mathbb{P} \text{ sctId};$ 
   $op : \text{expressionComparisonOperator}; subj\_or\_targets : \text{Sctids\_or\_Error} \bullet$ 
  i_attributeExpressionConstraint ss rf atts op subj_or_targets =
  if subj_or_targets  $\in \text{ran error}$ 
  then qerror (error ~ subj_or_targets)
  else if #rf = 0  $\wedge op = \text{eco\_eq}$  then
    quad_value({t : result_sctids subj_or_targets;
      a : atts; rels : ss.relationships |
      t_sctid ~ rels.t = t  $\wedge$  rels.a = a • rels}, source_direction)
  else if #rf = 1  $\wedge op = \text{eco\_eq}$  then
    quad_value({s : result_sctids subj_or_targets;
      a : atts; rels : ss.relationships |
      rels.s = s  $\wedge$  rels.t  $\in \text{ran } t\_sctid \wedge$  rels.a = a • rels}, targets_direction)
  else if #rf = 0  $\wedge op = \text{eco\_neq}$  then
    quad_value({t : result_sctids subj_or_targets;
      a : atts; rels : ss.relationships |
      t_sctid ~ rels.t  $\neq$  t  $\wedge$  rels.a = a • rels}, source_direction)
  else
    quad_value({s : result_sctids subj_or_targets;
      a : atts; rels : ss.relationships |
      rels.s  $\neq$  s  $\wedge$  rels.t  $\in \text{ran } t\_sctid \wedge$  rels.a = a • rels}, targets_direction)
```

4.2 concreteAttributeConstraint

$i_concreteAttributeConstraint :$ $Substrate \rightarrow \mathbb{P} \text{ sctId} \rightarrow concreteComparisonOperator \rightarrow$ $concreteValue \rightarrow Quads_or_Error$
$\forall ss : Substrate; \text{atts} : \mathbb{P} \text{ sctId}; \text{op} : concreteComparisonOperator;$ $val : concreteValue \bullet$ $i_concreteAttributeConstraint \text{ ss atts op val} =$ $quad_value(\{rels : ss.relationships \mid rels.a \in atts \wedge rels.t \in \text{ran } t_concrete \wedge val \in concreteMatch(t_co$
$concreteMatch :$ $concreteValue \rightarrow concreteComparisonOperator \rightarrow \mathbb{P} concreteValue$

Interpretation: Apply the substrate descendants (*descs*) or ancestors (*ancs*) function to a set of sctId's in the supplied *Sctids_or_Error*. Error conditions are propagated.

$i_constraintOperator :$ $Substrate \rightarrow constraintOperator[0 \dots 1] \rightarrow Sctids_or_Error \rightarrow Sctids_or_Error$ $completeFun : (sctId \rightarrow \mathbb{P} \text{ sctId}) \rightarrow sctId \rightarrow \mathbb{P} \text{ sctId}$
$\forall ss : Substrate; \text{oco} : constraintOperator[0 \dots 1]; \text{subresult} : Sctids_or_Error \bullet$ $i_constraintOperator \text{ ss oco subresult} =$ $\text{if } error \sim \text{subresult} \in ERROR \vee \#oco = 0$ then subresult $\text{else if head oco} = descendantOrSelfOf$ $\text{ then } ok(\bigcup \{id : result_sctids \text{ subresult} \bullet$ $completeFun \text{ ss.descendants } id\} \cup result_sctids \text{ subresult})$ $\text{else if head oco} = descendantOf$ $\text{ then } ok(\bigcup \{id : result_sctids \text{ subresult} \bullet$ $completeFun \text{ ss.descendants } id\})$ $\text{else if head oco} = ancestorOrSelfOf$ $\text{ then } ok(\bigcup \{id : result_sctids \text{ subresult} \bullet$ $completeFun \text{ ss.ancestors } id\} \cup result_sctids \text{ subresult})$ $\text{else } ok(\bigcup \{id : result_sctids \text{ subresult}$ $\bullet completeFun \text{ ss.ancestors } id\})$ $\forall f : (sctId \rightarrow \mathbb{P} \text{ sctId}); id : sctId \bullet completeFun f id =$ $\text{if } id \in \text{dom } f \text{ then } f \text{ id else } \emptyset$

4.3 FocusConcept

`focusConcept = [memberOf] conceptReference`

4.3.1 focusConcept

focusConcept is either a simple concept reference or the interpretation of the **memberOf** function applied to a concept reference.

$$\begin{aligned} \text{focusConcept} ::= & \\ & \text{focusConcept}_m \langle\langle \text{conceptReference} \rangle\rangle \mid \\ & \text{focusConcept}_c \langle\langle \text{conceptReference} \rangle\rangle \end{aligned}$$

Interpretation: If **memberOf** is present the interpretation of **focusConcept** is union the interpretation of **memberOf** applied to each element in the interpretation of **conceptReference**. If **memberOf** isn't part of the spec, the interpretation is the interpretation of **conceptReference** itself

$i_focusConcept : Substrate \rightarrow focusConcept \rightarrow Sctids_or_Error$
$\begin{aligned} \forall ss : Substrate; fc : focusConcept \bullet \\ i_focusConcept\ ss\ fc = \\ \quad \text{if } focusConcept_c \sim fc \in conceptReference \\ \quad \quad \text{then } ss.i_conceptReference\ (focusConcept_c \sim fc) \\ \quad \text{else } i_memberOf\ ss\ (ss.i_conceptReference\ (focusConcept_m \sim fc)) \end{aligned}$

4.3.2 memberOf

memberOf returns the union of the application of the substrate *refset* function to each of the supplied reference set identifiers. An error is returned if (a) *refsetids* already has an error or (b) one or more of the refset identifiers aren't substrate *refsetIds*

$i_memberOf : Substrate \rightarrow Sctids_or_Error \rightarrow Sctids_or_Error$
$\begin{aligned} \forall ss : Substrate; refsetids : Sctids_or_Error \bullet \\ i_memberOf\ ss\ refsetids = \\ \quad \text{if } refsetids \in \text{ran } error \\ \quad \quad \text{then } refsetids \\ \quad \text{else } bigunion\{sctid : result_sctids\ refsetids \bullet ss.i_refsetId\ sctid\} \end{aligned}$

5 Glue and Helper Functions

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 *Quads* or an error condition. If it is a collection of *Quads*, 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 *sctIds*.
- *target_direction* – matches were performed on attribute/source (**reverseFlag** was present). Return the target *sctIds* or, eventually, concrete values

$[noitemsep, nolistsep] direction ::= source_direction \mid targets_direction$
 $Quads_or_Error ::= quad_value \langle \mathbb{P} Quad \times direction \rangle \mid qerror \langle ERROR \rangle$

5.1.2 IDGroups

IDGroups represent carries a list of *sctIds* and the corresponding groups that they passed in, if successful, otherwise an error indication.

$IDGroups ::= group_value \langle \mathbb{P}(sctId \times groupId) \rangle \mid gerror \langle ERROR \rangle$

5.2 Result transformations

- **result_sctids** – the set of *sctIds* 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
- **to_idGroups** – the *sctId* to *groupId* part of in an id group or an empty map if there is error
- **quads_to_idgroups** – convert a set of quads int a set of id groups using the following rules:
 - If the set of quads has an error, propagate it
 - If the quad direction is *source_direction* (target to source) a list of unique relationship subjects and, for each subjects, the set of different groups it appears as a subject in
 - Otherwise return a list of relationship target sctids and, for each target, the set of different groups it appears as a target in.
- **idgroups_to_sctids** – remove the groups and return an *Sctids_or_Error* for the ids

$ \begin{aligned} & \text{result_sctids} : \text{Sctids_or_Error} \rightarrow \mathbb{P} \text{ sctId} \\ & \text{quads_for} : \text{Quads_or_Error} \rightarrow \mathbb{P} \text{ Quad} \\ & \text{quad_direction} : \text{Quads_or_Error} \rightarrow \text{direction} \\ & \text{to_idGroups} : \text{IDGroups} \rightarrow \text{sctId} \rightarrow \mathbb{P} \text{ groupId} \\ & \text{quads_to_idgroups} : \text{Quads_or_Error} \rightarrow \text{IDGroups} \\ & \text{idgroups_to_sctids} : \text{IDGroups} \rightarrow \text{Sctids_or_Error} \\ & \text{quads_to_sctids} : \text{Quads_or_Error} \rightarrow \text{Sctids_or_Error} \end{aligned} $
$ \begin{aligned} & \forall r : \text{Sctids_or_Error} \bullet \text{result_sctids } r = \\ & \quad \text{if } r \in \text{ran error then } \emptyset \\ & \quad \text{else } \text{ok} \sim r \\ & \forall q : \text{Quads_or_Error} \bullet \text{quads_for } q = \\ & \quad \text{if } q \in \text{ran qerror then } \emptyset \\ & \quad \text{else } \text{first} (\text{quad_value} \sim q) \\ & \forall q : \text{Quads_or_Error} \bullet \text{quad_direction } q = \\ & \quad \text{second} (\text{quad_value} \sim q) \\ & \forall g : \text{IDGroups}; \text{id} : \text{sctId} \bullet \text{to_idGroups } g \text{ id} = \\ & \quad \text{if } g \in \text{ran gerror then } \emptyset \\ & \quad \text{else } \{ge : (\text{group_value} \sim g) \mid \text{first } ge = \text{id} \bullet \text{second } ge\} \\ & \forall q : \text{Quads_or_Error} \bullet \text{quads_to_idgroups } q = \\ & \quad \text{if } q \in \text{ran qerror} \\ & \quad \quad \text{then } \text{gerror}(\text{qerror} \sim q) \\ & \quad \text{else if } \text{quad_direction } q = \text{source_direction} \\ & \quad \quad \text{then } \text{group_value}\{qe : \text{quads_for } q \bullet (qe.s, qe.g)\} \\ & \quad \text{else } \text{group_value}\{qe : \text{quads_for } q \bullet (t_sctid \sim qe.t, qe.g)\} \\ & \forall g : \text{IDGroups} \bullet \text{idgroups_to_sctids } g = \\ & \quad \text{if } g \in \text{ran gerror then } \text{ok } \emptyset \\ & \quad \text{else } \text{ok} (\text{dom}(\text{group_value} \sim g)) \\ & \forall q : \text{Quads_or_Error} \bullet \text{quads_to_sctids } q = \\ & \quad \text{if } q \in \text{ran qerror then } \text{error} (\text{qerror} \sim q) \\ & \quad \text{else } \text{ok } \{qe : \text{quads_for } q \bullet qe.s\} \end{aligned} $

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)
- **qfirstError** – convert two *Sctids_or_Error* types, into a *Quads_or_Error* propagating at least one of the errors. (not fully defined)
- **gfirstError** – convert two *IDGroups* into a single *IDGroups* 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.

$firstError : \mathbb{P} Sctids_or_Error \rightarrow Sctids_or_Error$ $qfirstError : \mathbb{P} Sctids_or_Error \rightarrow Quads_or_Error$ $gfirstError : \mathbb{P} IDGroups \rightarrow IDGroups$ $union, intersect, minus : Sctids_or_Error \rightarrow Sctids_or_Error \rightarrow Sctids_or_Error$ $bigunion, bigintersect : \mathbb{P} Sctids_or_Error \rightarrow Sctids_or_Error$ $gunion, gintersect, gminus : IDGroups \rightarrow IDGroups \rightarrow IDGroups$
$\forall x, y : Sctids_or_Error \bullet union\ x\ y =$ if $x \in \text{ran } error \vee y \in \text{ran } error$ then $firstError\ \{x, y\}$ else $ok\ ((ok \sim x) \cup (ok \sim y))$ $\forall x, y : Sctids_or_Error \bullet intersect\ x\ y =$ if $x \in \text{ran } error \vee y \in \text{ran } error$ then $firstError\ \{x, y\}$ else $ok\ ((ok \sim x) \cap (ok \sim y))$ $\forall x, y : Sctids_or_Error \bullet minus\ x\ y =$ if $x \in \text{ran } error \vee y \in \text{ran } error$ then $firstError\ \{x, y\}$ else $ok\ ((ok \sim x) \setminus (ok \sim y))$ $\forall rs : \mathbb{P} Sctids_or_Error \bullet bigunion\ rs =$ if $\exists r : rs \bullet r \in \text{ran } error$ then $firstError\ rs$ else $ok\ (\bigcup \{r : rs \bullet result_sctids\ r\})$ $\forall rs : \mathbb{P} Sctids_or_Error \bullet bigintersect\ rs =$ if $\exists r : rs \bullet r \in \text{ran } error$ then $firstError\ rs$ else $ok\ (\bigcap \{r : rs \bullet result_sctids\ r\})$ $\forall x, y : IDGroups \bullet gintersect\ x\ y =$ if $x \in \text{ran } gerror \vee y \in \text{ran } gerror$ then $gfirstError\ \{x, y\}$ else $group_value((group_value \sim x) \cap (group_value \sim y))$ $\forall x, y : IDGroups \bullet gunion\ x\ y =$ if $x \in \text{ran } gerror \vee y \in \text{ran } gerror$ then $gfirstError\ \{x, y\}$ else $group_value((group_value \sim x) \cup (group_value \sim y))$ $\forall x, y : IDGroups \bullet gminus\ x\ y =$ if $x \in \text{ran } gerror \vee y \in \text{ran } gerror$ then $gfirstError\ \{x, y\}$ else $group_value((group_value \sim x) \setminus (group_value \sim y))$

6 Appendix 1 – 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 $head\ T$.

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

7 Appendix 2 – Generic cardinality evaluation

evalCardinality Evaluate the cardinality of an arbitrary set of type T .

- If the cardinality isn't supplied ($\#opt_cardinality = 0$), return the set.
- If the number of elements is greater or equal to the minimum cardinality ($first(head\ opt_cardinality)$) then:
 - If the max cardinality is an integer ($num \sim second(head\ opt_cardinality)$) and it is greater than or equal to the number of elements or:
 - the max cardinality is not specified ($second(head\ opt_cardinality) = many$)
 return the set
- Otherwise return \emptyset

$[T]$
$evalCardinality : cardinality[0 \dots 1] \rightarrow \mathbb{P} T \rightarrow \mathbb{P} T$
$\forall opt_cardinality : cardinality[0 \dots 1]; t : \mathbb{P} T \bullet$ $evalCardinality\ opt_cardinality\ t =$ if $\#opt_cardinality = 0 \vee$ $(\#t \geq first(head\ opt_cardinality)) \wedge$ $(second(head\ opt_cardinality) = many \vee$ $num \sim (second(head\ opt_cardinality)) \geq \#t)$ then t else \emptyset

8 Appendix 3 - 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 structure of the form $T \times seq_1 T$

And returns $Sctids_or_Error$

In the formalization below, $first\ seq_e$ refers to the left hand side of the $T \times seq_1 T$ 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]	$ \begin{aligned} & \text{applyToSequence} : \text{Substrate} \rightarrow (\text{Substrate} \rightarrow T \rightarrow \text{Sctids_or_Error}) \rightarrow \\ & \quad (\text{Sctids_or_Error} \rightarrow \text{Sctids_or_Error} \rightarrow \text{Sctids_or_Error}) \rightarrow \\ & \quad (T \times \text{seq}_1 T) \rightarrow \text{Sctids_or_Error} \\ & \forall ss : \text{Substrate}; f : (\text{Substrate} \rightarrow T \rightarrow \text{Sctids_or_Error}); \\ & \quad op : (\text{Sctids_or_Error} \rightarrow \text{Sctids_or_Error} \rightarrow \text{Sctids_or_Error}); \\ & \quad seq_e : (T \times \text{seq}_1 T) \bullet \\ & \text{applyToSequence } ss \ f \ op \ seq_e = \\ & \text{if } tail(second \ seq_e) = \langle \rangle \text{ then} \\ & \quad op \ (f \ ss \ (first \ seq_e)) \ (f \ ss \ (head \ (second \ seq_e))) \\ & \text{else} \\ & \quad op \ (f \ ss \ (first \ seq_e)) \ (\text{applyToSequence } ss \ f \ op \ (head \ (second \ seq_e), tail \ (second \ seq_e))) \end{aligned} $
-----	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------