A Declarative Semantics for SNOMED CT Expression Constraints

April 2, 2015

Contents

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. 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 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.

$$
\begin{array}{|l}
is\_a : sctId \\
zero\_group : groupId \\
attribute\_concept : sctId \\
refset\_concept : sctId
\end{array}
$$

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

$$s : sctId$$
$$a : sctId$$
$$t : target$$
$$g : groupId$$

$$a = is\_a \Rightarrow (g = zero\_group \wedge t \in \operatorname{ran} t\_sctid)$$

2.1.1 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.*"

```
conceptReference = conceptId [ws "|" ws term ws "|"]
conceptId = sctId
attributeName = conceptReference
```

$$
\begin{aligned}
&\mathit{conceptReference} == \mathit{conceptId} \times \mathit{term}[0\,..\,1]\\
&\mathit{conceptId} == \mathit{sctId}\\
&\mathit{attributeName} == \mathit{conceptReference}
\end{aligned}
$$

2.2 Substrate

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

$$
\begin{array}{l}
concepts : \mathbb{P}\, sctId \\
relationships : \mathbb{P}\, Quad \\
parentsOf : sctId \nrightarrow \mathbb{P}\, sctId \\
equivalent\_concepts : sctId \nrightarrow \mathbb{P}\, sctId \\
refsets : sctId \nrightarrow \mathbb{P}\, sctId \\
\\
childrenOf : sctId \nrightarrow \mathbb{P}\, sctId \\
descendants : sctId \nrightarrow \mathbb{P}\, sctId \\
ancestors : sctId \nrightarrow \mathbb{P}\, sctId \\
\\
groups : \mathbb{P}\, groupId \\
subjects : \mathbb{P}\, sctId \\
targets : \mathbb{P}\, target \\
\\
i\_conceptReference : conceptReference \rightarrow Sctids\_or\_Error \\
i\_attributeName : attributeName \rightarrow Sctids\_or\_Error \\
i\_refsetId : sctId \rightarrow Sctids\_or\_Error
\end{array}
$$

$$
\begin{array}{l}
\forall\, rel : relationships \bullet rel.s \in concepts \land rel.a \in concepts \land \\
\qquad\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\qquad parentsOf\ c1 = parentsOf\ c2 \land \\
\qquad\qquad \{r : relationships \mid r.s = c1\} = \{r : relationships \mid r.s = c2\} \land \\
\qquad\qquad \{r : relationships \mid r.a = c1\} = \{r : relationships \mid r.a = c2\} \land \\
\qquad\qquad \{r : relationships \mid t\_sctid^{\sim} r.t = c1\} = \{r : relationships \mid t\_sctid^{\sim} r.t = c2\} \land \\
\qquad\qquad c1 \in \operatorname{dom} refsets \Leftrightarrow c2 \in \operatorname{dom} refsets \land \\
\qquad\qquad c1 \in \operatorname{dom} refsets \Rightarrow refsets\ c1 = refsets\ c2 \land \\
\qquad\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\qquad descendants\ s = childrenOf\ s \cup \bigcup\{t : childrenOf\ s \bullet descendants\ t\} \\
\forall\, t : concepts \bullet \\
\qquad\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\qquad result\_sctids\ cr\_interp \subseteq concepts \\
\forall\, att\_interp : \operatorname{ran} i\_attributeName \mid att\_interp \in \operatorname{ran} ok \bullet \\
\qquad\qquad result\_sctids\ att\_interp \subseteq concepts \\
\forall\, refset\_interp : \operatorname{ran} i\_refsetId \mid refset\_interp \in \operatorname{ran} ok \bullet \\
\qquad\qquad result\_sctids\ refset\_interp \subseteq concepts
\end{array}
$$

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

$$
\begin{array}{l}
\textit{strict\_substrate} \\
\hline
\textit{Substrate} \\
\hline
\forall\, cr : conceptReference \bullet i\_conceptReference\ cr = \\
\quad \textbf{if}\ \mathit{first}\ cr \notin concepts\ \textbf{then}\ error\ unknownConceptReference \\
\quad \textbf{else}\ ok\ (equivalent\_concepts\ (\mathit{first}\ cr)) \\
\\
\forall\, an : attributeName \bullet i\_attributeName\ an = \\
(\textbf{let}\ rslt == i\_conceptReference\ an \bullet \\
\quad \textbf{if}\ rslt \in \operatorname{ran} error\ \textbf{then}\ rslt \\
\quad \textbf{else if}\ result\_sctids\ rslt \subseteq (descendants\ attribute\_concept) \\
\quad\quad \textbf{then}\ rslt \\
\quad \textbf{else}\ error\ unknownAttributeId) \\
\\
\forall\, rsid : sctId \bullet i\_refsetId\ rsid = \\
\quad \textbf{if}\ rsid \in descendants\ refset\_concept \\
\quad\quad \textbf{then}\ ok\ \{rsid\} \\
\quad \textbf{else}\ error\ unknownRefsetId \\
\hline
\end{array}
$$

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*

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

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

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` .

$$\begin{aligned}&expressionConstraint ::=\\ &\quad expcons\_refined\langle\langle refinedExpressionConstraint\rangle\rangle \mid\\ &\quad expcons\_unrefined\langle\langle unrefinedExpressionConstraint\rangle\rangle\end{aligned}$$

$$\begin{aligned}&i\_expressionConstraint :\\ &\quad Substrate \to expressionConstraint \to Sctids\_or\_Error\\ \hline &\forall\, ss : Substrate;\ ec : expressionConstraint \bullet i\_expressionConstraint\ ss\ ec =\\ &\mathbf{if}\ ec \in \operatorname{ran} expcons\_refined\\ &\quad \mathbf{then}\ i\_refinedExpressionConstraint\ ss\ (expcons\_refined^{\sim}\, ec)\\ &\mathbf{else}\ i\_unrefinedExpressionConstraint\ ss\ (expcons\_unrefined^{\sim}\, ec)\end{aligned}$$

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`

$$
\begin{aligned}
&unrefinedExpressionConstraint ::= \\
&\qquad unrefined\_compound \langle\langle compoundExpressionConstraint \rangle\rangle \mid \\
&\qquad unrefined\_simple \langle\langle simpleExpressionConstraint \rangle\rangle
\end{aligned}
$$

$$
\begin{aligned}
&i\_unrefinedExpressionConstraint : \\
&\qquad Substrate \to unrefinedExpressionConstraint \to Sctids\_or\_Error \\
\hline
&\forall\, ss : Substrate;\ uec : unrefinedExpressionConstraint \bullet \\
&i\_unrefinedExpressionConstraint\ ss\ uec = \\
&\mathbf{if}\ uec \in \operatorname{ran} unrefined\_compound \\
&\quad \mathbf{then}\ i\_compoundExpressionConstraint\ ss\ (unrefined\_compound^{\sim} uec) \\
&\quad \mathbf{else}\ i\_simpleExpressionConstraint\ ss\ (unrefined\_simple^{\sim} uec)
\end{aligned}
$$

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.

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

$$
\begin{array}{l}
i\_refinedExpressionConstraint : \\
\quad Substrate \rightarrow refinedExpressionConstraint \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ rec : refinedExpressionConstraint \bullet \\
i\_refinedExpressionConstraint\ ss\ rec = \\
\quad intersect\ (i\_unrefinedExpressionConstraint\ ss\ (first\ rec))(i\_refinement\ ss\ (second\ rec))
\end{array}
$$

$$
\begin{array}{l}
i\_refinedExpressionConstraint' : \\
\quad Substrate \rightarrow refinedExpressionConstraint' \rightarrow Sctids\_or\_Error
\end{array}
$$

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.

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

$$
\begin{array}{l}
i\_simpleExpressionConstraint : \\
\qquad Substrate \rightarrow simpleExpressionConstraint \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ sec : simpleExpressionConstraint \bullet \\
i\_simpleExpressionConstraint\ ss\ sec = \\
\qquad i\_constraintOperator\ ss\ (first\ sec)\ (i\_focusConcept\ ss\ (second\ sec))
\end{array}
$$

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 / disjunctionExpressionConstraint / exclusionExpressionConstraint / "(" ws compoundExpressionConstraint ws ")"

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

$$
\begin{array}{l}
\mathit{compoundExpressionConstraint} ::= \\
\qquad \mathit{compound\_conj} \langle\langle \mathit{conjunctionExpressionConstraint} \rangle\rangle \mid \\
\qquad \mathit{compound\_disj} \langle\langle \mathit{disjunctionExpressionConstraint} \rangle\rangle \mid \\
\qquad \mathit{compound\_excl} \langle\langle \mathit{exclusionExpressionConstraint} \rangle\rangle \mid \\
\qquad \mathit{compound\_nested} \langle\langle \mathit{compoundExpressionConstraint}' \rangle\rangle \\
[\mathit{compoundExpressionConstraint}']
\end{array}
$$

$$
\begin{array}{l}
\mathit{i\_compoundExpressionConstraint} : \\
\qquad \mathit{Substrate} \to \mathit{compoundExpressionConstraint} \to \mathit{Sctids\_or\_Error} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ \mathit{cec} : \mathit{compoundExpressionConstraint} \bullet \\
\mathit{i\_compoundExpressionConstraint}\ \mathit{ss}\ \mathit{cec} = \\
\textbf{if}\ \mathit{cec} \in \mathrm{ran}\ \mathit{compound\_conj} \\
\quad \textbf{then}\ \mathit{i\_conjunctionExpressionConstraint}\ \mathit{ss}\ (\mathit{compound\_conj}^{\sim}\ \mathit{cec}) \\
\textbf{else if}\ \mathit{cec} \in \mathrm{ran}\ \mathit{compound\_disj} \\
\quad \textbf{then}\ \mathit{i\_disjunctionExpressionConstraint}\ \mathit{ss}\ (\mathit{compound\_disj}^{\sim}\ \mathit{cec}) \\
\textbf{else if}\ \mathit{cec} \in \mathrm{ran}\ \mathit{compound\_excl} \\
\quad \textbf{then}\ \mathit{i\_exclusionExpressionConstraint}\ \mathit{ss}\ (\mathit{compound\_excl}^{\sim}\ \mathit{cec}) \\
\textbf{else}\ \mathit{i\_compoundExpressionConstraint}'\ \mathit{ss}\ (\mathit{compound\_nested}^{\sim}\ \mathit{cec})
\end{array}
$$

$$
\begin{array}{l}
\mathit{i\_compoundExpressionConstraint}' : \\
\qquad \mathit{Substrate} \to \mathit{compoundExpressionConstraint}' \to \mathit{Sctids\_or\_Error}
\end{array}
$$

3.2.5 conjunctionExpressionConstraint

"A conjunction expression constraint combines two or more expression constraints 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 subExpressionConstraint) `conjunctionExpressionConstraint` is interpreted as the intersection of the interpretations of the `subExpressionConstraints` .

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

Apply the intersection operator to the interpretation of each subExpressionConstraint

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

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

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)
```

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

`disjunctionExpressionConstraint` is interpreted as the union 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` minus the interpretation of the second.

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

$$i\_exclusionExpressionConstraint : \\ \quad Substrate \to exclusionExpressionConstraint \to Sctids\_or\_Error$$

$$\forall ss : Substrate;\ ecr : exclusionExpressionConstraint \bullet \\ i\_exclusionExpressionConstraint\ ss\ ecr = \\ \quad minus\ (i\_subExpressionConstraint\ ss\ (first\ ecr))(i\_subExpressionConstraint\ ss\ (second\ ecr))$$

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 (compoundExpressionConstraint / refinedExpressionConstraint) ws ")"

The interpretation of `subExpressionConstraint` is the interpretation the `simpleExpressionConstraint` `compoundExpressionConstraint` or `refinedExpressionConstraint`

$$subExpressionConstraint ::= \\ \quad subExpr\_simple\langle\langle simpleExpressionConstraint\rangle\rangle \mid \\ \quad subExpr\_compound\langle\langle compoundExpressionConstraint'\rangle\rangle \mid \\ \quad subExpr\_refined\langle\langle refinedExpressionConstraint'\rangle\rangle$$

$$
\begin{array}{l}
i\_subExpressionConstraint : \\
\qquad Substrate \rightarrow subExpressionConstraint \rightarrow Sctids\_or\_Error \\
\hline
\forall ss : Substrate;\ sec : subExpressionConstraint \bullet \\
i\_subExpressionConstraint\ ss\ sec = \\
\mathbf{if}\ sec \in \operatorname{ran} subExpr\_simple \\
\quad \mathbf{then}\ i\_simpleExpressionConstraint\ ss\ (subExpr\_simple^{\sim} sec) \\
\mathbf{else\ if}\ sec \in \operatorname{ran} subExpr\_compound \\
\quad \mathbf{then}\ i\_compoundExpressionConstraint'\ ss\ (subExpr\_compound^{\sim} sec) \\
\mathbf{else}\ i\_refinedExpressionConstraint'\ ss\ (subExpr\_refined^{\sim} sec)
\end{array}
$$

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

$$
\begin{array}{l}
refinement == subRefinement \times refinementConjunctionOrDisjunction[0 \mathinner{.\,.} 1] \\
[refinement'] \\
refinementConjunctionOrDisjunction ::= \\
\quad refine\_conjset \langle\!\langle conjunctionRefinementSet \rangle\!\rangle\ | \\
\quad refine\_disjset \langle\!\langle disjunctionRefinementSet \rangle\!\rangle
\end{array}
$$

$$
\begin{array}{l}
i\_refinement : Substrate \rightarrow refinement \rightarrow Sctids\_or\_Error \\
\hline
\forall ss : Substrate;\ rfnment : refinement \bullet \\
i\_refinement\ ss\ rfnment = \\
(\mathbf{let}\ lhs == i\_subRefinement\ ss\ (first\ rfnment);\ rhs == second\ rfnment \bullet \\
\mathbf{if}\ \#rhs = 0 \\
\quad \mathbf{then}\ lhs \\
\mathbf{else\ if}\ (head\ rhs) \in \operatorname{ran} refine\_conjset \\
\quad \mathbf{then}\ intersect\ lhs\ (i\_conjunctionRefinementSet\ ss\ (refine\_conjset^{\sim}(head\ rhs))) \\
\mathbf{else}\ union\ lhs\ (i\_disjunctionRefinementSet\ ss\ (refine\_disjset^{\sim}(head\ rhs))))
\end{array}
$$

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

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)
```

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

The interpretation of a `conjunctionRefinementSet` is the <u>intersection</u> of the interpretation of the `subrefinements` .

$$\begin{array}{l}
i_conjunctionRefinementSet : \\
\quad Substrate \rightarrow conjunctionRefinementSet \rightarrow Sctids_or_Error \\
\hline
\forall\, ss : Substrate;\ conjset : conjunctionRefinementSet \bullet \\
i_conjunctionRefinementSet\ ss\ conjset = \\
\mathbf{if}\ tail\ conjset = \langle\rangle \\
\quad \mathbf{then}\ i_subRefinement\ ss\ (head\ conjset) \\
\mathbf{else} \\
\quad intersect\ (i_subRefinement\ ss\ (head\ conjset))\ (i_conjunctionRefinementSet\ ss\ (tail\ conjset))
\end{array}$$

3.3.2 disjunctionRefinementSet

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

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

The interpretation of a `disjunctionRefinementSet` is the <u>union</u> of the interpretation of the `subrefinements` .

$$\begin{array}{l}
i_disjunctionRefinementSet : \\
\quad Substrate \rightarrow disjunctionRefinementSet \rightarrow Sctids_or_Error \\
\hline
\forall\, ss : Substrate;\ disjset : disjunctionRefinementSet \bullet \\
i_disjunctionRefinementSet\ ss\ disjset = \\
\mathbf{if}\ tail\ disjset = \langle\rangle \\
\quad \mathbf{then}\ i_subRefinement\ ss\ (head\ disjset) \\
\mathbf{else} \\
\quad union\ (i_subRefinement\ ss\ (head\ disjset))\ (i_disjunctionRefinementSet\ ss\ (tail\ disjset))
\end{array}$$

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*** .

$$
\begin{aligned}
&subRefinement ::= \\
&\quad subrefine\_attset \langle\langle attributeSet \rangle\rangle \mid \\
&\quad subrefine\_attgroup \langle\langle attributeGroup \rangle\rangle \mid \\
&\quad subrefine\_refinement \langle\langle refinement' \rangle\rangle
\end{aligned}
$$

$$
\begin{aligned}
&i\_subRefinement : \\
&\qquad Substrate \rightarrow subRefinement \rightarrow Sctids\_or\_Error \\
\hline
&\forall\, ss : Substrate;\ subrefine : subRefinement \bullet \\
&i\_subRefinement\ ss\ subrefine = \\
&\textbf{if}\ subrefine \in \operatorname{ran} subrefine\_attset \\
&\quad \textbf{then}\ i\_attributeSet\ ss\ (subrefine\_attset^{\sim}\, subrefine) \\
&\textbf{else if}\ subrefine \in \operatorname{ran} subrefine\_attgroup \\
&\quad \textbf{then}\ i\_attributeGroup\ ss\ (subrefine\_attgroup^{\sim}\, subrefine) \\
&\textbf{else}\ i\_refinement'\ ss\ (subrefine\_refinement^{\sim}\, subrefine)
\end{aligned}
$$

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

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

$$[attributeSet']$$

$$\begin{aligned}
&conjunctionOrDisjunctionAttributeSet ::= \\
&\quad attset_conjattset\langle\!\langle conjunctionAttributeSet\rangle\!\rangle \mid \\
&\quad attset_disjattset\langle\!\langle disjunctionAttributeSet\rangle\!\rangle
\end{aligned}$$

$$\begin{aligned}
&i_attributeSet : \\
&\quad Substrate \to attributeSet \to Sctids_or_Error \\
\hline
&\forall\, ss : Substrate;\ attset : attributeSet \bullet \\
&i_attributeSet\ ss\ attset = \\
&(\mathbf{let}\ lhs == i_subAttributeSet\ ss\ (first\ attset);\ rhs == second\ attset \bullet \\
&\mathbf{if}\ \#rhs = 0 \\
&\quad \mathbf{then}\ lhs \\
&\mathbf{else\ if}\ head\ rhs \in \operatorname{ran} attset_conjattset \\
&\quad \mathbf{then}\ intersect\ lhs\ (i_conjunctionAttributeSet\ ss\ (attset_conjattset^{\sim}(head\ rhs))) \\
&\mathbf{else}\ union\ lhs\ (i_disjunctionAttributeSet\ ss\ (attset_conjattset^{\sim}(head\ rhs))))
\end{aligned}$$

$$\begin{aligned}
&i_attributeSet' : \\
&\quad Substrate \to attributeSet' \to Sctids_or_Error
\end{aligned}$$

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` .

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

$$\begin{aligned}
&i_conjunctionAttributeSet : \\
&\quad Substrate \to conjunctionAttributeSet \to Sctids_or_Error \\
\hline
&\forall\, ss : Substrate;\ conjset : conjunctionAttributeSet \bullet \\
&i_conjunctionAttributeSet\ ss\ conjset = \\
&\mathbf{if}\ tail\ conjset = \langle\rangle \\
&\quad \mathbf{then}\ i_subAttributeSet\ ss\ (head\ conjset) \\
&\mathbf{else} \\
&\quad intersect\ (i_subAttributeSet\ ss\ (head\ conjset))\ (i_conjunctionAttributeSet\ ss\ (tail\ conjset))
\end{aligned}$$

3.4.2 disjunctionAttributeSet

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

```
disjunctionAttributeSet = 1*(ws disjunction ws subAttributeSet)
```

The interpretation of a *disjunctionAttributeSet* is the union of the interpretations of its *subAttributeSets* .

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

$$\begin{array}{l}
i_disjunctionAttributeSet : \\
\quad Substrate \rightarrow disjunctionAttributeSet \rightarrow Sctids_or_Error \\
\hline
\forall\, ss : Substrate;\ disjset : disjunctionAttributeSet \bullet \\
i_disjunctionAttributeSet\ ss\ disjset = \\
\mathbf{if}\ tail\ disjset = \langle\rangle \\
\quad \mathbf{then}\ i_subAttributeSet\ ss\ (head\ disjset) \\
\mathbf{else} \\
\quad union\ (i_subAttributeSet\ ss\ (head\ disjset))\ (i_disjunctionAttributeSet\ ss\ (tail\ disjset))
\end{array}$$

3.4.3 subAttributeSet

> *"A subAttributeSet is either an attribute or a bracketed attribute set."*

```
subAttributeSet = attribute / "(" ws attributeSet ws ")"
```

The interpretation of a *subAttributeSet* is either the interpretation of the *attribute* or the interpretation of the *attributeSet* .

$$\begin{array}{l}
subAttributeSet ::= \\
\quad subaset_attribute\langle\langle attribute\rangle\rangle \mid \\
\quad subaset_attset\langle\langle attributeSet'\rangle\rangle
\end{array}$$

$$\begin{array}{l}
i_subAttributeSet : \\
\qquad Substrate \rightarrow subAttributeSet \rightarrow Sctids_or_Error \\
\hline
\forall\, ss : Substrate;\ subaset : subAttributeSet \bullet \\
i_subAttributeSet\ ss\ subaset = \\
\mathbf{if}\ subaset \in \mathrm{ran}\ subaset_attribute \\
\quad \mathbf{then}\ i_attribute\ ss\ (subaset_attribute^{\sim} subaset) \\
\mathbf{else}\ i_attributeSet'\ ss\ (subaset_attset^{\sim} subaset)
\end{array}$$

3.4.4 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.

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

$$attributeOperatorValue ::= attrib\_expr \langle\langle expressionComparisonOperator \times expressionConstraintValue \rangle\rangle \mid$$

$attrib\_num\langle\langle numericComparisonOperator \times numericValue\rangle\rangle \mid$

$attrib\_str\langle\langle stringComparisonOperator \times stringValue\rangle\rangle$

$numericValue ::= nv\_decimal\langle\langle decimalValue\rangle\rangle \mid nv\_integer\langle\langle \mathbb{N}\rangle\rangle$

$[reverseFlag]$

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

3.4.5 Attribute Cardinality Interpretation

> *"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) "]"
```

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

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 $= 0$

$$
\begin{array}{l}
i\_att\_cardinality : \\
\quad Substrate \rightarrow cardinality[0\,..\,1] \rightarrow Quads\_or\_Error \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ ocard : cardinality[0\,..\,1];\ qore : Quads\_or\_Error \bullet \\
i\_att\_cardinality\ ss\ ocard\ qore = \\
\textbf{if}\ qore \in \operatorname{ran} qerror \\
\quad \textbf{then}\ error(qerror^{\sim}\, qore) \\
\textbf{else if}\ \#ocard = 0 \\
\quad \textbf{then}\ ok\,(i\_required\_cardinality\ 1\ many\ qore) \\
\textbf{else if}\ first\,(head\ ocard) > 0 \\
\quad \textbf{then}\ ok\,(i\_required\_cardinality\,(first\,(head\ ocard))\,(second\,(head\ ocard))\ qore) \\
\textbf{else}\ ok\,(i\_optional\_cardinality\ ss\,(second\,(head\ ocard))\ qore)
\end{array}
$$

$i\_required\_cardinality$ returns the set of subject or target sctids that meet the cardinality requirements

$$
\begin{array}{l}
i\_required\_cardinality : \\
\quad \mathbb{N} \rightarrow unlimitedNat \rightarrow Quads\_or\_Error \rightarrow \mathbb{P}\, sctId \\
\hline
\forall\, min : \mathbb{N};\ max : unlimitedNat;\ qore : Quads\_or\_Error \bullet \\
i\_required\_cardinality\ min\ max\ qore = \\
\textbf{if}\ quad\_direction\ qore = source\_direction \\
\textbf{then}\ \{subj : \{q : quads\_for\ qore \bullet q.s\} \mid \\
\quad evalCardinality[Quad]\ min\ max\ \{q : quads\_for\ qore \mid q.s = subj\} \neq \emptyset\} \\
\textbf{else}\ \{targ : \{q : quads\_for\ qore \bullet t\_sctid^{\sim}\, q.t\} \mid \\
\quad evalCardinality[Quad]\ min\ max\ \{q : quads\_for\ qore \mid (t\_sctid^{\sim}\, q.t) = targ\} \neq \emptyset\}
\end{array}
$$

$i\_optional\_cardinality$ 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.

$$
\begin{array}{|l}
\hline
i\_optional\_cardinality : \\
\quad Substrate \rightarrow unlimitedNat \rightarrow Quads\_or\_Error \ \rightarrow \mathbb{P}\, sctId \\
\hline
\forall\, ss : Substrate;\ max : unlimitedNat;\ qore : Quads\_or\_Error \bullet \\
i\_optional\_cardinality\ ss\ max\ qore = \\
\mathbf{if}\ quad\_direction\ qore = source\_direction \\
\mathbf{then} \\
\quad ss.concepts \setminus (\{q : quads\_for\ qore \bullet q.s\} \setminus (i\_required\_cardinality\, 0\, max\ qore)) \\
\mathbf{else} \\
\quad ss.concepts \setminus (\{q : quads\_for\ qore \bullet t\_sctid^{\sim} q.t\} \setminus (i\_required\_cardinality\, 0\, max\ qore)) \\
\hline
\end{array}
$$

3.4.6 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`

$$
\begin{array}{l}
expressionConstraintValue ::= \\
\quad expression\_simple \langle\!\langle simpleExpressionConstraint \rangle\!\rangle \mid \\
\quad expression\_refined \langle\!\langle refinedExpressionConstraint' \rangle\!\rangle \mid \\
\quad expression\_compound \langle\!\langle compoundExpressionConstraint \rangle\!\rangle
\end{array}
$$

$$
\begin{array}{|l}
\hline
i\_expressionConstraintValue : \\
\quad Substrate \rightarrow expressionConstraintValue \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ ecv : expressionConstraintValue \bullet \\
i\_expressionConstraintValue\ ss\ ecv = \\
\mathbf{if}\ ecv \in \operatorname{ran} expression\_simple \\
\quad \mathbf{then}\ i\_simpleExpressionConstraint\ ss\ (expression\_simple^{\sim} ecv) \\
\mathbf{else\ if}\ ecv \in \operatorname{ran} expression\_refined \\
\quad \mathbf{then}\ i\_refinedExpressionConstraint'\ ss\ (expression\_refined^{\sim} ecv) \\
\mathbf{else}\ i\_compoundExpressionConstraint\ ss\ (expression\_compound^{\sim} ecv) \\
\hline
\end{array}
$$

3.4.7 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.*"

```
expressionComparisonOperator = "=" / "!="
```

The interpretation of *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 *expressionConstraintValue*

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

$$
\begin{array}{l}
i\_expressionComparisonOperator : \\
\quad Substrate \rightarrow reverseFlag[0 \mathrel{..} 1] \rightarrow \mathbb{P}\, sctId \rightarrow \\
\quad (expressionComparisonOperator \times expressionConstraintValue) \rightarrow Quads\_or\_Error \\
\hline
\forall\, ss : Substrate;\ rf : reverseFlag[0 \mathrel{..} 1];\ atts : \mathbb{P}\, sctId; \\
\quad ec : (expressionComparisonOperator \times expressionConstraintValue) \bullet \\
i\_expressionComparisonOperator\ ss\ rf\ atts\ ec = \\
(\mathbf{let}\ ecv == i\_expressionConstraintValue\ ss\ (second\ ec) \bullet \\
\mathbf{if}\ ecv \in \operatorname{ran} error\ \mathbf{then}\ qerror(error^{\sim} ecv) \\
\mathbf{else\ if}\ \#reverseFlag = 0 \land (first\ ec = eco\_eq) \\
\quad \mathbf{then}\ quad\_value(\{q : ss.relationships \mid q.t \in \operatorname{ran} t\_sctid \land (t\_sctid^{\sim} q.t) \in (result\_sctids\ ecv)\}, \\
\qquad\quad source\_direction) \\
\mathbf{else\ if}\ \#reverseFlag > 0 \land (first\ ec = eco\_eq) \\
\quad \mathbf{then}\ quad\_value(\{q : ss.relationships \mid q.s \in (result\_sctids\ ecv)\}, \\
\qquad\quad targets\_direction) \\
\mathbf{else\ if}\ \#reverseFlag = 0 \land (first\ ec = eco\_neq) \\
\quad \mathbf{then}\ quad\_value(\{q : ss.relationships \mid q.t \in \operatorname{ran} t\_sctid \land (t\_sctid^{\sim} q.t) \notin (result\_sctids\ ecv)\}, \\
\qquad\quad source\_direction) \\
\mathbf{else}\ quad\_value(\{q : ss.relationships \mid q.s \notin (result\_sctids\ ecv)\}, \\
\qquad\quad targets\_direction))
\end{array}
$$

3.4.8 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.*"

numericComparisonOperator = "=" / "!=" / "<=" / "<" / ">=" / ">"

The interpretation of `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

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

$$\begin{array}{l}
i_numericComparisonOperator : \\
\quad Substrate \rightarrow reverseFlag[0 \mathrel{..} 1] \rightarrow \mathbb{P}\, sctId \rightarrow \\
\quad (numericComparisonOperator \times numericValue) \rightarrow Quads_or_Error \\
\hline
\forall\, ss : Substrate;\ rf : reverseFlag[0 \mathrel{..} 1];\ atts : \mathbb{P}\, sctId; \\
\quad ncv : (numericComparisonOperator \times numericValue) \bullet \\
i_numericComparisonOperator\ ss\ rf\ atts\ ncv = \\
\textbf{if}\ \#rf > 0\ \textbf{then} \\
\quad quad_value(\emptyset, targets_direction) \\
\textbf{else} \\
\quad quad_value(\{q : ss.relationships \mid q.a \in atts \land q.t \in \operatorname{ran} t_concrete \land \\
numericComparison\,(t_concrete^{\sim}\, q.t)\,(first\ ncv) = (second\ ncv)\}, source_direction)
\end{array}$$

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"

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

3.4.9 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 = "=" / "!="

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

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

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

$$
\begin{array}{l}
stringComparison : \\
\quad concreteValue \to stringComparisonOperator \nrightarrow stringValue
\end{array}
$$

3.5 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 "}"
```

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

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 *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*

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

$$
\begin{array}{l}
i\_attributeGroup : Substrate \rightarrow attributeGroup \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ ag : attributeGroup \bullet \\
i\_attributeGroup\ ss\ ag = \\
\quad i\_group\_cardinality\ (first\ ag)\ (i\_groupAttributeSet\ ss\ (second\ ag))
\end{array}
$$

3.5.1 Attribute Group Cardinality Interpretation

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 >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

$$
\begin{array}{l}
i\_group\_cardinality : \\
\quad Substrate \rightarrow cardinality[0\mathrel{..}1] \rightarrow sctIdGroups\_or\_Error \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ ocard : cardinality[0\mathrel{..}1];\ idg : sctIdGroups\_or\_Error \bullet \\
i\_group\_cardinality\ ss\ ocard\ idg = \\
\mathbf{if}\ idq \in \operatorname{ran} gerror \\
\quad \mathbf{then}\ error(gerror^{\sim} idg) \\
\mathbf{else}\ (\mathbf{let}\ gvals == group\_value^{\sim} idg \bullet \\
\quad \mathbf{if}\ \#ocard = 0 \\
\quad\quad \mathbf{then}\ ok(i\_required\_group\_cardinality\ 1\ many\ gvals) \\
\quad \mathbf{else\ if}\ first\ ocard > 0 \\
\quad\quad \mathbf{then}\ ok(i\_required\_group\_cardinality\ (first\ ocard)\ (second\ ocard)\ gvals) \\
\quad \mathbf{else}\ ok(i\_optional\_group\_cardinality\ ss\ (second\ ocard)\ gvals))
\end{array}
$$

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

$$
\begin{array}{l}
i\_required\_group\_cardinality : \\
\quad \mathbb{N} \rightarrow unlimitedNat \rightarrow \mathbb{P}\, sctIdGroup \rightarrow \mathbb{P}\, sctId \\
\hline
\forall\, min : \mathbb{N};\ max : unlimitedNat;\ groups : \mathbb{P}\, sctIdGroup \bullet \\
i\_required\_group\_cardinality\ min\ max\ groups = \\
\{sctid : \{s : groups \bullet first\ s\} \mid \\
\quad evalCardinality[sctIdGroup]\ min\ max \{g : groups \mid first\ g = sctid\} \neq \emptyset\}
\end{array}
$$

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.

$$
\begin{array}{l}
i\_optional\_group\_cardinality : \\
\quad Substrate \to unlimitedNat \to \mathbb{P}\, sctIdGroup \to \mathbb{P}\, sctid \\
\hline
\forall\, ss : Substrate;\ max : unlimitedNat;\ groups : \mathbb{P}\, sctIdGroup \bullet \\
i\_optional\_group\_cardinality\ ss\ max\ groups = \\
ss.concepts \setminus \\
\quad (\{s : groups \bullet first\ s\} \setminus (i\_required\_group\_cardinality\ 0\ max\ groups))
\end{array}
$$

3.5.2 attributeSet for 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.

$$
\begin{array}{l}
i\_groupAttributeSet : \\
\quad Substrate \to attributeSet \to sctIdGroups\_or\_Error \\
\hline
\forall\, ss : Substrate;\ attset : attributeSet \bullet \\
i\_groupAttributeSet\ ss\ attset = \\
(\textbf{let}\ lhs == i\_groupSubAttributeSet\ ss\ (first\ attset);\ rhs == second\ attset \bullet \\
\textbf{if}\ \#rhs = 0 \\
\quad \textbf{then}\ lhs \\
\textbf{else if}\ head\ rhs \in \operatorname{ran} attset\_conjattset \\
\quad \textbf{then}\ gintersect\ lhs\ (i\_groupConjunctionAttributeSet\ ss\ (attset\_conjattset^{\sim}(head\ rhs))) \\
\textbf{else}\ gunion\ lhs\ (i\_groupDisjunctionAttributeSet\ ss\ (attset\_conjattset^{\sim}(head\ rhs))))
\end{array}
$$

3.5.3 conjunctionAttributeSet for Groups

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

$$
\begin{array}{l}
i\_groupConjunctionAttributeSet : \\
\quad Substrate \to conjunctionAttributeSet \to sctIdGroups\_or\_Error \\
\hline
\forall\, ss : Substrate;\ conjset : conjunctionAttributeSet \bullet \\
i\_groupConjunctionAttributeSet\ ss\ conjset = \\
\textbf{if}\ tail\ conjset = \langle\rangle \\
\quad \textbf{then}\ i\_groupSubAttributeSet\ ss\ (head\ conjset) \\
\textbf{else} \\
\quad gintersect\ (i\_groupSubAttributeSet\ ss\ (head\ conjset))\ (i\_groupConjunctionAttributeSet\ ss\ (tail\ conj
\end{array}
$$

3.5.4 disjunctionAttributeSet for Groups

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*.

$$
\begin{array}{l}
\mathit{i\_groupDisjunctionAttributeSet} : \\
\quad \mathit{Substrate} \rightarrow \mathit{groupDisjunctionAttributeSet} \rightarrow \mathit{sctIdGroups\_or\_Error} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ \mathit{disjset} : \mathit{disjunctionAttributeSet} \bullet \\
\mathit{i\_groupConjunctionAttributeSet}\ \mathit{ss}\ \mathit{disjset} = \\
\mathbf{if}\ \mathit{tail}\ \mathit{disjset} = \langle\rangle \\
\quad \mathbf{then}\ \mathit{i\_groupSubAttributeSet}\ \mathit{ss}\ (\mathit{head}\ \mathit{disjset}) \\
\mathbf{else} \\
\quad \mathit{gunion}\ (\mathit{i\_groupSubAttributeSet}\ \mathit{ss}\ (\mathit{head}\ \mathit{disjset}))\ (\mathit{i\_groupConjunctionAttributeSet}\ \mathit{ss}\ (\mathit{tail}\ \mathit{disjset}))
\end{array}
$$

3.5.5 subAttributeSet for Groups

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*.

$$
\begin{array}{l}
\mathit{i\_groupSubAttributeSet} : \\
\qquad \mathit{Substrate} \rightarrow \mathit{subAttributeSet} \rightarrow \mathit{sctIdGroups\_or\_Error} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ \mathit{subaset} : \mathit{subAttributeSet} \bullet \\
\mathit{i\_groupSubAttributeSet}\ \mathit{ss}\ \mathit{subaset} = \\
\mathbf{if}\ \mathit{subaset} \in \mathrm{ran}\ \mathit{subaset\_attribute} \\
\quad \mathbf{then}\ \mathit{i\_groupAttribute}\ \mathit{ss}\ (\mathit{subaset\_attribute}^{\sim}\ \mathit{subaset}) \\
\mathbf{else}\ \mathit{i\_groupAttributeSet}'\ \mathit{ss}\ (\mathit{subaset\_attset}^{\sim}\ \mathit{subaset})
\end{array}
$$

$$
\begin{array}{l}
\mathit{i\_groupSubAttributeSet} : \\
\qquad \mathit{Substrate} \rightarrow \mathit{subAttributeSet} \rightarrow \mathit{sctIdGroups\_or\_Error}
\end{array}
$$

3.5.6 attribute for Groups

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*).

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

3.6 Group Attribute Cardinality

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

- Only non-zero groups are evaluated.
- A sctid can pass in the context of more than one group

i_att_group_cardinality takes an optional cardinality and a set of quads and returns a list sctids and the groups in which they passed the cardinality constraint.

$$
\begin{array}{l}
i\_att\_group\_cardinality : \\
\quad Substrate \rightarrow cardinality[0\,..\,1] \rightarrow Quads\_or\_Error \rightarrow sctIdGroups\_or\_Error \\
\hline
\forall\, ss : Substrate;\; ocard : cardinality[0\,..\,1];\; qore : Quads\_or\_Error \bullet \\
i\_att\_group\_cardinality\ ss\ ocard\ qore = \\
\textbf{if}\ qore \in \operatorname{ran} qerror \\
\quad \textbf{then}\ gerror(qerror^{\sim} qore) \\
\textbf{else if}\ \#ocard = 0 \\
\quad \textbf{then}\ group\_value\,(i\_required\_att\_group\_cardinality\ 1\ many\ qore) \\
\textbf{else if}\ first\,(head\ ocard) > 0 \\
\quad \textbf{then}\ group\_value\,(i\_required\_att\_group\_cardinality\,(first\,(head\ ocard))\,(second\,(head\ ocard))\ qore) \\
\textbf{else} \\
\quad group\_value\,(i\_optional\_att\_group\_cardinality\ ss\,(second\,(head\ ocard))\ qore)
\end{array}
$$

i_required_att_group_cardinality returns the set of subject or target sctids combined with the group(s) in which they qualified

$$
\begin{array}{l}
i\_required\_att\_group\_cardinality : \\
\quad \mathbb{N} \rightarrow unlimitedNat \rightarrow Quads\_or\_Error \rightarrow \mathbb{P}\, sctIdGroup \\
\hline
\forall\, min : \mathbb{N};\; max : unlimitedNat;\; qore : Quads\_or\_Error \bullet \\
i\_required\_att\_group\_cardinality\ min\ max\ qore = \\
\textbf{if}\ quad\_direction\ qore = source\_direction \\
\textbf{then} \\
\quad \{subjgroup : \{q : quads\_for\ qore \mid q.g \neq zero\_group \bullet (q.s, q.g)\} \mid \\
\quad evalCardinality[Quad]\ min\ max\ \{q : quads\_for\ qore \mid \\
\quad\quad q.s = first\ subjgroup \wedge q.g = second\ subjgroup\} \neq \emptyset\} \\
\textbf{else}\ \{targgroup : \{q : quads\_for\ qore \mid q.g \neq zero\_group \wedge q.t \in \operatorname{ran} t\_sctid \bullet (t\_sctid^{\sim} q.t, q.g)\} \mid \\
\quad evalCardinality[Quad]\ min\ max\ \{q : quads\_for\ qore \mid \\
\quad\quad t\_sctid^{\sim} q.t = first\ targgroup \wedge q.g = second\ targgroup\} \neq \emptyset\}
\end{array}
$$

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.

$$
\begin{array}{l}
i\_optional\_att\_group\_cardinality : \\
\quad Substrate \rightarrow unlimitedNat \rightarrow Quads\_or\_Error \rightarrow \mathbb{P}\, sctIdGroup \\
\hline
\forall\, ss : Substrate;\ max : unlimitedNat;\ qore : Quads\_or\_Error \bullet \\
i\_optional\_att\_group\_cardinality\ ss\ max\ qore = \\
\mathbf{if}\ quad\_direction\ qore = source\_direction \\
\mathbf{then} \\
\quad \{rel : ss.relationships \mid rel.g \neq zero\_group \bullet (rel.s, rel.g)\} \setminus \\
\quad \{q : quads\_for\ qore \mid q.g \neq zero\_group \bullet (q.s, q.g)\} \cup \\
\qquad i\_required\_att\_group\_cardinality\ 0\ max\ qore \\
\mathbf{else} \\
\quad \{rel : ss.relationships \mid rel.g \neq zero\_group \wedge rel.t \in \operatorname{ran} t\_sctid \bullet (t\_sctid^{\sim} rel.t, rel.g)\} \setminus \\
\quad \{q : quads\_for\ qore \mid q.g \neq zero\_group \bullet (t\_sctid^{\sim} q.t, q.g)\} \cup \\
\qquad i\_required\_att\_group\_cardinality\ 0\ max\ qore
\end{array}
$$

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 $constraintOperator[0 \mathrel{..} 1]$ 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 ($\emptyset$).

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

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

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

$$\begin{array}{l}
\forall\, ss : Substrate;\ oco : constraintOperator[0\,..\,1];\ input : Sctids_or_Error \bullet \\
i_constraintOperator\ ss\ oco\ input = \\
\quad \mathbf{if}\ error^{\sim} input \in ERROR \vee \#oco = 0 \\
\quad\quad \mathbf{then}\ input \\
\quad \mathbf{else\ if}\ head\ oco = descendantOrSelfOf \\
\quad\quad \mathbf{then}\ ok(\bigcup\{id : result_sctids\ input \bullet \\
\quad\quad\quad\quad completeFun\ ss.descendants\ id\} \cup result_sctids\ input) \\
\quad \mathbf{else\ if}\ head\ oco = descendantOf \\
\quad\quad \mathbf{then}\ ok(\bigcup\{id : result_sctids\ input \bullet \\
\quad\quad\quad\quad completeFun\ ss.descendants\ id\,\}) \\
\quad \mathbf{else\ if}\ head\ oco = ancestorOrSelfOf \\
\quad\quad \mathbf{then}\ ok(\bigcup\{id : result_sctids\ input \bullet \\
\quad\quad\quad\quad completeFun\ ss.ancestors\ id\} \cup result_sctids\ input) \\
\quad \mathbf{else}\ ok(\bigcup\{id : result_sctids\ input \\
\quad\quad\quad\quad \bullet\ completeFun\ ss.ancestors\ id\,\}) \\
\\
\forall\, f : (sctId \rightarrowtail \mathbb{P}\, sctId);\ id : sctId \bullet completeFun\ f\ id = \\
\quad \mathbf{if}\ id \in \operatorname{dom} f\ \mathbf{then}\ f\ id\ \mathbf{else}\ \emptyset
\end{array}$$

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 FocusConcept

> *"A focus concept is a concept reference, 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."*

```
focusConcept = [memberOf] conceptReference
```

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

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 specified, the interpretation is the substrate interpretations of `conceptReference` itself

$$
\begin{array}{l}
\mathit{i\_focusConcept} : \mathit{Substrate} \rightarrow \mathit{focusConcept} \rightarrow \mathit{Sctids\_or\_Error} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ \mathit{fc} : \mathit{focusConcept} \bullet \\
\mathit{i\_focusConcept}\ \mathit{ss}\ \mathit{fc} = \\
\quad \textbf{if}\ \mathit{focusConcept\_c}^{\sim}\mathit{fc} \in \mathit{conceptReference} \\
\quad \textbf{then} \\
\qquad \mathit{ss}.\mathit{i\_conceptReference}\,(\mathit{focusConcept\_c}^{\sim}\mathit{fc}) \\
\quad \textbf{else} \\
\qquad \mathit{i\_memberOf}\ \mathit{ss}\,(\mathit{ss}.\mathit{i\_conceptReference}\,(\mathit{focusConcept\_m}^{\sim}\mathit{fc}))
\end{array}
$$

4.3.1 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) the substrate interpretation of a given refsetId returns an error.

$$
\begin{array}{l}
\mathit{i\_memberOf} : \mathit{Substrate} \rightarrow \mathit{Sctids\_or\_Error} \rightarrow \mathit{Sctids\_or\_Error} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ \mathit{refsetids} : \mathit{Sctids\_or\_Error} \bullet \\
\mathit{i\_memberOf}\ \mathit{ss}\ \mathit{refsetids} = \\
\quad \textbf{if}\ \mathit{refsetids} \in \mathrm{ran}\ \mathit{error} \\
\qquad \textbf{then}\ \mathit{refsetids} \\
\quad \textbf{else}\ \mathit{bigunion}\{\mathit{sctid} : \mathit{result\_sctids}\ \mathit{refsetids} \bullet \mathit{ss}.\mathit{i\_refsetId}\ \mathit{sctid}\}
\end{array}
$$

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

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

5.1.2 sctIdGroups

A *sctIdGroup* is the combination of an *sctId* and a non-zero *groupId*. The set of all possbile *sctIdGroups* in a given substrate would be the union of all relationship subjects combined with the non-zero relationship groups in which they appear in the substrate relationships.

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

$$
\begin{aligned}
&\mathit{sctIdGroup} == \mathit{sctId} \times \mathit{groupId} \\
&\mathit{sctIdGroups\_or\_Error} ::= \mathit{group\_value}\langle\!\langle \mathbb{P}\, \mathit{sctIdGroup} \rangle\!\rangle \mid \\
&\qquad \mathit{gerror}\langle\!\langle \mathit{ERROR} \rangle\!\rangle
\end{aligned}
$$

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

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

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.

$$
\begin{array}{l}
\mathit{firstError} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \rightarrow \mathit{Sctids\_or\_Error} \\
\mathit{gfirstError} : \mathbb{P}\, \mathit{sctIdGroups\_or\_Error} \rightarrow \mathit{sctIdGroups\_or\_Error} \\[4pt]
\mathit{union}, \mathit{intersect}, \mathit{minus} : \mathit{Sctids\_or\_Error} \rightarrow \mathit{Sctids\_or\_Error} \rightarrow \\
\qquad \mathit{Sctids\_or\_Error} \\
\mathit{bigunion}, \mathit{bigintersect} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \rightarrow \mathit{Sctids\_or\_Error} \\[4pt]
\mathit{gunion}, \mathit{gintersect}, \mathit{gminus} : \mathit{sctIdGroups\_or\_Error} \rightarrow \mathit{sctIdGroups\_or\_Error} \rightarrow \mathit{sctIdGroups\_or\_Error} \\
\hline
\forall x, y : \mathit{Sctids\_or\_Error} \bullet \mathit{union}\ x\ y = \\
\quad \mathbf{if}\ x \in \operatorname{ran} \mathit{error} \lor y \in \operatorname{ran} \mathit{error}\ \mathbf{then}\ \mathit{firstError}\ \{x, y\} \\
\quad \mathbf{else}\ \mathit{ok}\ ((\mathit{ok}^{\sim} x) \cup (\mathit{ok}^{\sim} y)) \\[4pt]
\forall x, y : \mathit{Sctids\_or\_Error} \bullet \mathit{intersect}\ x\ y = \\
\quad \mathbf{if}\ x \in \operatorname{ran} \mathit{error} \lor y \in \operatorname{ran} \mathit{error}\ \mathbf{then}\ \mathit{firstError}\ \{x, y\} \\
\quad \mathbf{else}\ \mathit{ok}\ ((\mathit{ok}^{\sim} x) \cap (\mathit{ok}^{\sim} y)) \\[4pt]
\forall x, y : \mathit{Sctids\_or\_Error} \bullet \mathit{minus}\ x\ y = \\
\quad \mathbf{if}\ x \in \operatorname{ran} \mathit{error} \lor y \in \operatorname{ran} \mathit{error}\ \mathbf{then}\ \mathit{firstError}\ \{x, y\} \\
\quad \mathbf{else}\ \mathit{ok}\ ((\mathit{ok}^{\sim} x) \setminus (\mathit{ok}^{\sim} y)) \\[4pt]
\forall \mathit{rs} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \bullet \mathit{bigunion}\ \mathit{rs} = \\
\quad \mathbf{if}\ \exists r : \mathit{rs} \bullet r \in \operatorname{ran} \mathit{error}\ \mathbf{then}\ \mathit{firstError}\ \mathit{rs} \\
\quad \mathbf{else}\ \mathit{ok}\ (\bigcup \{r : \mathit{rs} \bullet \mathit{result\_sctids}\ r\}) \\[4pt]
\forall \mathit{rs} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \bullet \mathit{bigintersect}\ \mathit{rs} = \\
\quad \mathbf{if}\ \exists r : \mathit{rs} \bullet r \in \operatorname{ran} \mathit{error}\ \mathbf{then}\ \mathit{firstError}\ \mathit{rs} \\
\quad \mathbf{else}\ \mathit{ok}\ (\bigcap \{r : \mathit{rs} \bullet \mathit{result\_sctids}\ r\}) \\[4pt]
\forall x, y : \mathit{sctIdGroups\_or\_Error} \bullet \mathit{gintersect}\ x\ y = \\
\quad \mathbf{if}\ x \in \operatorname{ran} \mathit{gerror} \lor y \in \operatorname{ran} \mathit{gerror}\ \mathbf{then}\ \mathit{gfirstError}\ \{x, y\} \\
\quad \mathbf{else}\ \mathit{group\_value}((\mathit{group\_value}^{\sim} x) \cap (\mathit{group\_value}^{\sim} y)) \\[4pt]
\forall x, y : \mathit{sctIdGroups\_or\_Error} \bullet \mathit{gunion}\ x\ y = \\
\quad \mathbf{if}\ x \in \operatorname{ran} \mathit{gerror} \lor y \in \operatorname{ran} \mathit{gerror}\ \mathbf{then}\ \mathit{gfirstError}\ \{x, y\} \\
\quad \mathbf{else}\ \mathit{group\_value}((\mathit{group\_value}^{\sim} x) \cup (\mathit{group\_value}^{\sim} y)) \\[4pt]
\forall x, y : \mathit{sctIdGroups\_or\_Error} \bullet \mathit{gminus}\ x\ y = \\
\quad \mathbf{if}\ x \in \operatorname{ran} \mathit{gerror} \lor y \in \operatorname{ran} \mathit{gerror}\ \mathbf{then}\ \mathit{gfirstError}\ \{x, y\} \\
\quad \mathbf{else}\ \mathit{group\_value}((\mathit{group\_value}^{\sim} x) \setminus (\mathit{group\_value}^{\sim} y))
\end{array}
$$

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 $\mathit{head}\ T$.

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

7 Appendix 2 – Generic cardinality evaluation

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

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

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 tuple of the form "$T \times \mathrm{seq}_1\, T$"

And returns *Sctids_or_Error*

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

$$
\begin{array}{l}
[T] \\
\hline
\mathit{applyToSequence} : \mathit{Substrate} \to (\mathit{Substrate} \to T \to \mathit{Sctids\_or\_Error}) \to \\
\qquad (\mathit{Sctids\_or\_Error} \to \mathit{Sctids\_or\_Error} \to \mathit{Sctids\_or\_Error}) \to \\
\qquad (T \times \mathrm{seq}_1\ T) \to \mathit{Sctids\_or\_Error} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ f : (\mathit{Substrate} \to T \to \mathit{Sctids\_or\_Error}); \\
\qquad \mathit{op} : (\mathit{Sctids\_or\_Error} \to \mathit{Sctids\_or\_Error} \to \mathit{Sctids\_or\_Error}); \\
\qquad \mathit{seq\_e} : (T \times \mathrm{seq}_1\ T) \bullet \\
\mathit{applyToSequence}\ \mathit{ss}\ f\ \mathit{op}\ \mathit{seq\_e} = \\
\textbf{if}\ \mathit{tail}(\mathit{second}\ \mathit{seq\_e}) = \langle\rangle\ \textbf{then} \\
\qquad \mathit{op}\ (f\ \mathit{ss}\ (\mathit{first}\ \mathit{seq\_e}))(f\ \mathit{ss}\ (\mathit{head}\ (\mathit{second}\ \mathit{seq\_e}))) \\
\textbf{else} \\
\qquad \mathit{op}\ (f\ \mathit{ss}\ (\mathit{first}\ \mathit{seq\_e}))(\mathit{applyToSequence}\ \mathit{ss}\ f\ \mathit{op}\ (\mathit{head}\ (\mathit{second}\ \mathit{seq\_e}), \mathit{tail}\ (\mathit{second}\ \mathit{seq\_e})))
\end{array}
$$