A Declarative Semantics for SNOMED CT Expression Constraints

March 9, 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, specifically:

- **SCTID** – a SNOMED CT identifier
- **TERM** – a fully specified name, preferred term or synonym for a SNOMED CT Concept
- **REAL** – a real number
- **STRING** – a string literal
- **GROUP** – a role group identifier
- $\mathbb{N}$ – a non-negative integer
- $\mathbb{Z}$ – an integer

$$[SCTID, TERM, REAL, STRING, GROUP]$$

We will also need to recognize some well known identifiers: the *is_a* attribute, the *zero_group* and *attribute_concept*, the parent of all attributes

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

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 real literal
- **TARGET** – the target of a relationship that is either an *SCTID* or a *CONCRETEVALUE*

$$
\begin{array}{l}
CONCRETEVALUE ::= string\langle\langle STRING\rangle\rangle \mid integer\langle\langle \mathbb{Z}\rangle\rangle \mid real\langle\langle REAL\rangle\rangle \\
TARGET ::= object\langle\langle SCTID\rangle\rangle \mid concrete\langle\langle CONCRETEVALUE\rangle\rangle
\end{array}
$$

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*)

$$
\begin{array}{|l}
\hline
\textit{Quad} \\
s : SCTID \\
a : SCTID \\
t : TARGET \\
g : GROUP \\
\hline
a = is\_a \Rightarrow (g = zero\_group \wedge object^{\sim} t \in SCTID) \\
\hline
\end{array}
$$

2.2 Substrate

A substrate consists of:

- **concepts** The set of *SCTID*s (concepts) that are considered valid in the context of the substrate. *References to any SCTID that is not a member of concepts MUST be treated as an error.*
- **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
- **attributeIds** The descendantsOf the attribute_concept, including equivalents
- **refsetsIds** The descendants of the refset_concept, including equivalents

The formal definition of substrate follows, where c and r are given and the remainder are derived. The expressions below assert that:

1. All sources, attributes and SCTID targets of *relationships* are included in the substrate *concepts* list.
2. There is a *parentsOf* entry for every concept in the substrate *concepts* list.
3. Every sctid in the range of the *parentsOf* function is in the substrate *concepts* list.
4. Every *is_a* relationship entry is represented in the *parentsOf* function. (Note that the reverse isn't necessarily true).
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. 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
8. Every refset is a substrate *concepts*
9. Every member of a refset is a substrate *concept*
10. *childrenOf* is the inverse of *parentsOf*, where any concept that isn't a parent has no children.
11. *descendants* is the transitive closure of the *childrenOf* function
12. *ancestors* is the transitive closure of the *parentsOf* function
13. No concept can be its own ancestor (or, by inference, descendant)
14. Every *attributeId* is a substrate *concept*
15. Every *refsetId* is a substrate *concept*

$$
\begin{array}{l}
\textit{Substrate} \\
\hline
\mathit{concepts} : \mathbb{P}\, \mathit{SCTID} \\
\mathit{relationships} : \mathbb{P}\, \mathit{Quad} \\
\mathit{parentsOf} : \mathit{SCTID} \nrightarrow \mathbb{P}\, \mathit{SCTID} \\
\mathit{equivalent\_concepts} : \mathit{SCTID} \nrightarrow \mathbb{P}\, \mathit{SCTID} \\
\mathit{refsets} : \mathit{SCTID} \nrightarrow \mathbb{P}\, \mathit{SCTID} \\[4pt]
\mathit{childrenOf} : \mathit{SCTID} \nrightarrow \mathbb{P}\, \mathit{SCTID} \\
\mathit{descendants} : \mathit{SCTID} \nrightarrow \mathbb{P}\, \mathit{SCTID} \\
\mathit{ancestors} : \mathit{SCTID} \nrightarrow \mathbb{P}\, \mathit{SCTID} \\
\mathit{attributeIds} : \mathbb{P}\, \mathit{SCTID} \\
\mathit{refsetIds} : \mathbb{P}\, \mathit{SCTID} \\
\hline
\forall\, \mathit{rel} : \mathit{relationships} \bullet \mathit{rel}.s \in \mathit{concepts} \land \mathit{rel}.a \in \mathit{concepts} \land \\
\qquad\qquad (\mathit{object}^{\sim}\, \mathit{rel}.t \in \mathit{SCTID} \Rightarrow \mathit{object}^{\sim}\, \mathit{rel}.t \in \mathit{concepts}) \\[4pt]
\mathrm{dom}\, \mathit{parentsOf} = \mathit{concepts} \\
\bigcup(\mathrm{ran}\, \mathit{parentsOf}) \subseteq \mathit{concepts} \\[4pt]
\forall\, r : \mathit{relationships} \bullet r.a = \mathit{is\_a} \Rightarrow (\mathit{object}^{\sim}\, r.t) \in \mathit{parentsOf}\ r.a \\[4pt]
\mathrm{dom}\, \mathit{equivalent\_concepts} = \mathit{concepts} \\
\forall\, c : \mathit{concepts} \bullet c \in \mathit{equivalent\_concepts}\ c \\
\forall\, c1, c2 : \mathit{concepts} \mid c2 \in (\mathit{equivalent\_concepts}\ c1) \bullet \\
\qquad\qquad \mathit{parentsOf}\ c1 = \mathit{parentsOf}\ c2 \land \\
\qquad\qquad \{r : \mathit{relationships} \mid r.s = c1\} = \{r : \mathit{relationships} \mid r.s = c2\} \land \\
\qquad\qquad \{r : \mathit{relationships} \mid r.a = c1\} = \{r : \mathit{relationships} \mid r.a = c2\} \land \\
\qquad\qquad \{r : \mathit{relationships} \mid \mathit{object}^{\sim}\, r.t = c1\} = \{r : \mathit{relationships} \mid \mathit{object}^{\sim}\, r.t = c2\} \land \\
\qquad\qquad c1 \in \mathrm{dom}\, \mathit{refsets} \Leftrightarrow c2 \in \mathrm{dom}\, \mathit{refsets} \land \\
\qquad\qquad c1 \in \mathrm{dom}\, \mathit{refsets} \Rightarrow \mathit{refsets}\ c1 = \mathit{refsets}\ c2 \land \\
\qquad\qquad (\forall\, \mathit{rsd} : \mathrm{ran}\, \mathit{refsets} \bullet c1 \in \mathit{rsd} \Leftrightarrow c2 \in \mathit{rsd}) \\[4pt]
\mathrm{dom}\, \mathit{refsets} \subseteq \mathit{concepts} \\
\bigcup(\mathrm{ran}\, \mathit{refsets}) \subseteq \mathit{concepts} \\[4pt]
\mathrm{dom}\, \mathit{childrenOf} = \mathit{concepts} \\
\forall\, s, t : \mathit{concepts} \bullet t \in \mathit{parentsOf}\ s \Leftrightarrow s \in \mathit{childrenOf}\ t \\
\forall\, c : \mathit{concepts} \mid c \notin \bigcup(\mathrm{ran}\, \mathit{childrenOf}) \bullet \mathit{childrenOf}\ c = \emptyset \\[4pt]
\forall\, s : \mathit{concepts} \bullet \\
\qquad\qquad \mathit{descendants}\ s = \mathit{childrenOf}\ s \cup \bigcup\{t : \mathit{childrenOf}\ s \bullet \mathit{descendants}\ t\} \\
\forall\, t : \mathit{concepts} \bullet \\
\qquad\qquad \mathit{ancestors}\ t = \mathit{parentsOf}\ t \cup \bigcup\{s : \mathit{parentsOf}\ t \bullet \mathit{ancestors}\ s\} \\
\forall\, t : \mathit{concepts} \bullet t \notin \mathit{ancestors}\ t \\[4pt]
\mathit{attributeIds} \subseteq \mathit{concepts} \\
\mathit{refsetIds} \subseteq \mathit{concepts} \\
\hline
\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:

- There is at least one *SCTID* that is not substrate concept
- Every *attributeId* must be a descendant of *attribute_concept*
- Every *refsetId* must be a descendant of *refset_concept*
- *relationship* attributes must be *attributeId*s
- *refset* domains must be *refsetId*s

$$
\begin{array}{|l}
\hline
\textit{strict\_substrate} \\
\textit{Substrate} \\
\hline
\textit{concepts} \subset \textit{SCTID} \\
\textit{attributeIds} = \textit{descendants}\ \textit{attribute\_concept} \\
\forall\, r : \textit{relationships} \bullet r.a \in \textit{attributeIds} \\
\\
\textit{refsetIds} = \textit{descendants}\ \textit{refset\_concept} \\
\operatorname{dom} \textit{refsets} \subseteq \textit{refsetIds} \\
\hline
\end{array}
$$

2.2.3 Permissive Substrate

A permissive substrate is a substrate where every query will return some result – all $SCTID's$ are considered valid.

This includes the following rules:

1. Every possible *SCTID* is a substrate concept, attribute and a valid refset
2. The refset function will return a (possibly empty) set of results for any refuted

$$
\begin{array}{|l}
\hline
\textit{permissive\_substrate} \\
\textit{Substrate} \\
\hline
\textit{concepts} = \textit{SCTID} \wedge \textit{attributeIds} = \textit{concepts} \wedge \textit{refsetIds} = \textit{concepts} \\
\hline
\end{array}
$$

3 SCTIDS or Error Return

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*

$$\textit{ERROR} ::= \textit{unknownConceptReference} \mid \textit{unknownAttribute} \mid \textit{unknownRefsetId}$$

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

4 Interpretation of Expression Constraints

This section defines the interpretation of all language constructs that are interpreted in terms of other language constructs. 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.

4.1 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.

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

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

4.1.1 unrefinedExpressionConstraint

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

unrefinedExpressionConstraint = compoundExpressionConstraint / simpleExpressionConstraint

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

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

4.1.2 refinedExpressionConstraint

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

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.

$$\begin{array}{l} refinedExpressionConstraint == \\ \qquad unrefinedExpressionConstraint \times refinement \end{array}$$

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

4.1.3 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
```

$$simpleExpressionConstraint == constraintOperator[0 \mathinner{\ldotp\ldotp} 1] \times focusConcept$$

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

4.1.4 compoundExpressionConstraint

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

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

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

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

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

$$
\begin{array}{|l}
\hline
i\_compoundExpressionConstraint' : \\
\qquad Substrate \rightarrow compoundExpressionConstraint \rightarrow Sctids\_or\_Error \\
\hline
\end{array}
$$

4.1.5 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)
```

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

In the formalization below, *first cecr* refers to the left hand side of the *conjunctionExpressionConstraint* and *second cecr* to the right hand side. $head(second\ cecr)$ refers to the first element in the sequence and $tail(second\ cecr)$ refers to the remaining elements in the sequence, which may be empty ($\langle\rangle$).

$$
\begin{array}{l}
i\_conjunctionExpressionConstraint : \\
\quad Substrate \rightarrow conjunctionExpressionConstraint \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ cecr : conjunctionExpressionConstraint \bullet \\
i\_conjunctionExpressionConstraint\ ss\ cecr = \\
\mathbf{if}\ tail(second\ cecr) = \langle\rangle\ \mathbf{then} \\
\quad intersect\ (i\_subExpressionConstraint\ ss\ (first\ cecr)) \\
\qquad (i\_subExpressionConstraint\ ss\ (head\ (second\ cecr))) \\
\mathbf{else} \\
\quad intersect\ (i\_subExpressionConstraint\ ss\ (first\ cecr)) \\
\qquad (i\_conjunctionExpressionConstraint\ ss\ ((head(second\ cecr)) \\
\qquad\quad (tail(second\ cecr))))
\end{array}
$$

4.1.6 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)
```

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

In the formalization below, *first cecr* refers to the left hand side of the *disjunctionExpressionConstraint* and *second cecr* to the right hand side. $head(second\ cecr)$ refers to the first element in the sequence and $tail(second\ cecr)$ refers to the remaining elements in the sequence, which may be empty ($\langle\rangle$).

$$
\begin{array}{|l}
\hline
i\_disjunctionExpressionConstraint : \\
\qquad Substrate \to disjunctionExpressionConstraint \to Sctids\_or\_Error \\
\hline
\forall ss : Substrate;\ cecr : disjunctionExpressionConstraint \bullet \\
i\_disjunctionExpressionConstraint\ ss\ cecr = \\
\textbf{if}\ tail(second\ cecr) = \langle\rangle\ \textbf{then} \\
\quad union\ (i\_subExpressionConstraint\ ss\ (first\ cecr)) \\
\qquad\quad (i\_subExpressionConstraint\ ss\ (head\ (second\ cecr))) \\
\textbf{else} \\
\quad intersect\ (i\_subExpressionConstraint\ ss\ (first\ cecr)) \\
\qquad (i\_conjunctionExpressionConstraint\ ss\ ((head(second\ cecr)) \\
\qquad\quad (tail(second\ cecr)))) \\
\hline
\end{array}
$$

4.1.7 exclusionExpressionConstraint

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

exclusionExpressionConstraint = subExpressionConstraint ws exclusion ws subExpressionConstraint

$$
\begin{array}{l}
exclusionExpressionConstraint == \\
\qquad subExpressionConstraint \times subExpressionConstraint
\end{array}
$$

$$
\begin{array}{|l}
\hline
i\_exclusionExpressionConstraint : \\
\qquad Substrate \to exclusionExpressionConstraint \to Sctids\_or\_Error \\
\hline
\forall ss : Substrate;\ ecr : exclusionExpressionConstraint \bullet \\
i\_exclusionExpressionConstraint\ ss\ ecr = \\
\quad minus\ (i\_subExpressionConstraint\ ss\ (first\ ecr)) \\
\qquad\quad (i\_subExpressionConstraint\ ss\ (second\ ecr)) \\
\hline
\end{array}
$$

4.1.8 subExpressionConstraint

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

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

$$
\begin{array}{l}
subExpressionConstraint ::= \\
\quad subExpr\_simple \langle\langle simpleExpressionConstraint \rangle\rangle \mid \\
\quad subExpr\_compound \langle\langle compoundExpressionConstraint \rangle\rangle \mid \\
\quad subExpr\_refined \langle\langle refinedExpressionConstraint \rangle\rangle
\end{array}
$$

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

4.2 refinement

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

```
refinement = subRefinement / conjunctionGroup / disjunctionGroup
```

$$
\begin{array}{l}
refinement ::= \\
\quad refine\_subrefine \langle\langle subrefinement \rangle\rangle \mid \\
\quad refine\_conjg \langle\langle conjunctionGroup \rangle\rangle \mid \\
\quad refine\_disjg \langle\langle disjunctionGroup \rangle\rangle
\end{array}
$$

$$
\begin{array}{|l}
i\_refinement : Substrate \rightarrow refinement \rightarrow Sctids\_or\_Error \\
\hline
\forall ss : Substrate;\ rfnment : refinement \bullet i\_refinement = \\
\mathbf{if}\ refnment \in \operatorname{ran} refine\_subrefine \\
\quad \mathbf{then}\ i\_subrefinement\ ss\ (refine\_subrefine^{\sim}\ refnment) \\
\mathbf{else\ if}\ refnment \in \operatorname{ran} refine\_conjg \\
\quad \mathbf{then}\ i\_conjunctionGroup\ ss\ (refine\_conjg^{\sim}\ refnment) \\
\mathbf{else}\ i\_disjunctionGroup\ ss\ (refine\_disjg^{\sim}\ refnment)
\end{array}
$$

4.2.1 conjunctionGroup

```
conjunctionGroup = subRefinement 1*(conjunction subRefinement)
```

$$conjunctionGroup == \\ \quad subRefinement \times \mathrm{seq}_1(subRefinement)$$

$$\begin{array}{l}
i_conjunctionGroup : \\
\quad Substrate \rightarrow conjunctionGroup \rightarrow Sctids_or_Error \\
\hline
\forall\, ss : Substrate;\ conjg : conjunctionGroup \bullet \\
i_conjunctionGroup\ ss\ conjg = \\
\mathbf{if}\ tail(second\ conjg) = \langle\rangle\ \mathbf{then} \\
\quad intersect\ (i_subRefinement\ ss\ (first\ conjg)) \\
\qquad\quad (i_subRefinement\ ss\ (head\ (second\ conjg))) \\
\mathbf{else} \\
\quad intersect\ (i_subRefinement\ ss\ (first\ conjg)) \\
\qquad (i_conjunctionGroup\ ss\ ((head(second\ conjg)) \\
\qquad\quad (tail(second\ conjg))))
\end{array}$$

4.2.2 disjunctionGroup

```
disjunctionGroup = subRefinement 1*(disjunction subRefinement)
```

$$disjunctionGroup == \\ \quad subRefinement \times \mathrm{seq}_1(subRefinement)$$

$$\begin{array}{l}
i_disjunctionGroup : \\
\quad Substrate \rightarrow disjunctionGroup \rightarrow Sctids_or_Error \\
\hline
\forall\, ss : Substrate;\ disjg : disjunctionGroup \bullet \\
ii_disjunctionGroup\ ss\ disjg = \\
\mathbf{if}\ tail(second\ disjg) = \langle\rangle\ \mathbf{then} \\
\quad intersect\ (i_subRefinement\ ss\ (first\ disjg)) \\
\qquad\quad (i_subRefinement\ ss\ (head\ (second\ disjg))) \\
\mathbf{else} \\
\quad intersect\ (i_subRefinement\ ss\ (first\ disjg)) \\
\qquad (ii_disjunctionGroup\ ss\ ((head(second\ disjg)) \\
\qquad\quad (tail(second\ disjg))))
\end{array}$$

4.2.3 subRefinement

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

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

$$
\begin{array}{l}
\mathit{subRefinement} ::= \\
\quad \mathit{subrefine\_attset} \langle \mathit{attributeSet} \rangle \mid \\
\quad \mathit{subrefine\_attgroup} \langle \mathit{attributeGroup} \rangle \mid \\
\quad \mathit{subrefine\_refinement} \langle \mathit{refinement} \rangle
\end{array}
$$

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

4.3 attributeSet

```
attributeSet = subAttributeSet / conjunctionAttributeSet / disjunctionAttributeSet
```

$$
\begin{array}{l}
\mathit{attributeSet} ::= \\
\quad \mathit{attset\_subattset} \langle\!\langle \mathit{subAttributeSet} \rangle\!\rangle \mid \\
\quad \mathit{attset\_conjattset} \langle\!\langle \mathit{conjunctionAttributeSet} \rangle\!\rangle \mid \\
\quad \mathit{attset\_disjattset} \langle\!\langle \mathit{disjunctionAttributeSet} \rangle\!\rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{i\_attributeSet} : \\
\qquad \mathit{Substrate} \rightarrow \mathit{attributeSet} \rightarrow \mathit{Sctids\_or\_Error} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ \mathit{attset} : \mathit{attributeSet} \bullet \\
\mathbf{if}\ \mathit{attset} \in \operatorname{ran} \mathit{attset\_subattset} \\
\quad \mathbf{then}\ \mathit{i\_subAttributeSet}\ \mathit{ss}\ (\mathit{attset\_subattset}^{\sim}\ \mathit{attset}) \\
\mathbf{else\ if}\ \mathit{attset} \in \operatorname{ran} \mathit{attset\_conjattset} \\
\quad \mathbf{then}\ \mathit{i\_conjunctionAttributeSet}\ \mathit{ss}\ (\mathit{attset\_conjattset}^{\sim}\ \mathit{attset}) \\
\mathbf{else}\ \mathit{i\_disjunctionAttributeSet}\ \mathit{ss}\ (\mathit{attset\_disjattset}^{\sim}\ \mathit{attset})
\end{array}
$$

4.3.1 conjunctionAttributeSet

```
conjunctionAttributeSet = subAttributeSet 1*(conjunction subAttributeSet)
```

$$conjunctionAttributeSet == \\ \quad subAttributeSet \times \mathrm{seq}_1(subAttributeSet)$$

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

$$\begin{array}{l}
\forall ss : Substrate;\ conjaset : conjunctionAttributeSet \bullet \\
i_conjunctionAttributeSet\ ss\ conjaset = \\
\textbf{if}\ tail(second\ conjaset) = \langle\rangle\ \textbf{then} \\
\quad intersect\ (i_subAttributeSet\ ss\ (first\ conjaset)) \\
\qquad\quad (i_subAttributeSet\ ss\ (head\ (second\ conjaset))) \\
\textbf{else} \\
\quad intersect\ (i_subAttributeSet\ ss\ (first\ conjaset)) \\
\qquad (i_conjunctionGroup\ ss\ ((head(second\ conjaset)) \\
\qquad\quad (tail(second\ conjaset))))
\end{array}$$

4.3.2 disjunctionAttributeSet

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

$$disjunctionAttributeSet == \\ \quad subAttributeSet \times \mathrm{seq}_1(subAttributeSet)$$

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

$$\begin{array}{l}
\forall ss : Substrate;\ disjaset : disjunctionAttributeSet \bullet \\
i_disjunctionAttributeSet\ ss\ disjaset = \\
\textbf{if}\ tail(second\ disjaset) = \langle\rangle\ \textbf{then} \\
\quad intersect\ (i_subAttributeSet\ ss\ (first\ disjaset)) \\
\qquad\quad (i_subAttributeSet\ ss\ (head\ (second\ disjaset))) \\
\textbf{else} \\
\quad intersect\ (i_subAttributeSet\ ss\ (first\ disjaset)) \\
\qquad (i_disjunctionAttributeSet\ ss\ ((head(second\ disjaset)) \\
\qquad\quad (tail(second\ disjaset))))
\end{array}$$

4.3.3 subAttributeSet

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

$$
\begin{aligned}
&subAttributeSet == \\
&\quad subaset\_attribute\langle\!\langle attribute\rangle\!\rangle \mid \\
&\quad subaset\_attset\langle\!\langle attributeSet\rangle\!\rangle
\end{aligned}
$$

$$
\begin{array}{l}
i\_subAttributeSet : \\
\qquad\qquad Substrate \rightarrow subAttributeSet \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ subaset : subAttributeSet \bullet \\
\mathbf{if}\ subaset \in \operatorname{ran} subaset\_attribute \\
\quad \mathbf{then}\ i\_attribute\ ss\ (subaset\_attribute^{\sim}\, subaset) \\
\mathbf{else}\ i\_attributeSet\ ss\ (subaset\_attset^{\sim}\, subaset)
\end{array}
$$

4.4 attributeGroup

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

4.5 attribute

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

$$
\begin{aligned}
&attribute ::= \\
&\quad attrib\_conc\langle\!\langle concreteAttribute\rangle\!\rangle \mid \\
&\quad attrib\_expr\langle\!\langle expressionAttribute\rangle\!\rangle
\end{aligned}
$$

$$
\begin{array}{l}
i\_attribute : \\
\quad Substrate \rightarrow attribute \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ att : attribute \bullet \\
\mathbf{if}\ att \in \operatorname{ran} attrib\_conc \\
\quad \mathbf{then}\ i\_concreteAttribute\ ss\ (attrib\_conc^{\sim}\, att) \\
\mathbf{else}\ i\_expressionAttribute\ ss\ (attrib\_expr^{\sim}\, att)
\end{array}
$$

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

$$
\begin{aligned}
&unlimitedNat ::= num\langle\!\langle \mathbb{N}\rangle\!\rangle \mid many \\
&cardinality == \mathbb{N} \times unlimitedNat \\
&[reverseFlag]
\end{aligned}
$$

4.5.1 expressionAttribute

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

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

$$\begin{array}{|l}
\textit{expressionAttribute} \\
\hline
card : cardinality[0 \,..\, 1] \\
reverse : reverseFlag[0 \,..\, 1] \\
name : attributeName \\
operator : expressionComparisonOperator \\
value : expressionConstraintValue \\
\hline
\end{array}$$

4.5.2 concreteAttribute

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

$$\begin{aligned}
&concreteComparisonOperator ::= \\
&\qquad cco_eq \mid cco_neq \mid cco_leq \mid ccl_lt \mid cco_geq \mid cco_gt \\
&concreteValue ::= stringValue \mid integerValue \mid decimalValue
\end{aligned}$$

$$\begin{array}{|l}
\textit{concreteAttribute} \\
\hline
card : cardinality[0 \,..\, 1] \\
name : attributeName \\
operator : concreteComparisonOperator \\
value : concreteValue \\
\hline
\end{array}$$

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.

$$\begin{array}{|l}
i_concreteAttribute : \\
\qquad Substrate \rightarrow concreteAttribute \rightarrow Quads_or_Error \\
\hline
\forall\, ss : Substrate;\ ca : concreteAttribute \bullet \\
i_concreteAttribute\ ss\ ca = \\
(\mathbf{let}\ attids = i_attributeName\ ss\ ca.name \bullet \\
\qquad i_concreteAttributeConstraint\ ss\ attids\ ca.operator\ ca.value \\
\hline
\end{array}$$

The interpretation of an attribute. `attributeOperator` and `attributeName` determines the set of possible attributes in the substrate relationship table. `reverseFlag` and `expressionConstraintValue` determine the set of candidate targets (if `reverseFlag` is absent) or `source_direction` (if `reverseFlag` is present).

`cardinality` determines the minimum and maximum matches. In all cases, only a subset of the sources (targets if `reverseFlag` is present) in the substrate relationship table will be returned in the interpretation.

4.6 AttributeSubject

i_attributeSubject interprets the constraintOperator, if any in the context of the attribute name and interprets it as a set of *ATTRIBUTE*s

$$attributeOperator == \{descendantOrSelfOf, descendantOf\}$$
$$attributeSubject == attributeOperator[0\,..\,1] \times attributeName$$

$$\begin{array}{l} i\_attributeSubject : Substrate \nrightarrow attributeSubject \nrightarrow Sctids\_or\_Error \\ \hline \forall\, ss : Substrate;\; as : attributeSubject \bullet i\_attributeSubject\; ss\; as = \\ \quad i\_constraintOperator\; ss\; (first\; as)\; (i\_attributeName\; ss\; (second\; as)) \end{array}$$

4.7 Attribute

`attribute` consists of an optional `cardinality` and an `attributeConstraint`.

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

$$\begin{array}{l} \textit{attribute} \\ \hline card : cardinality[0\,..\,1] \\ attw : attributeConstraint \end{array}$$

attributeConstraint is either an attribute expression constraint or a concrete value constraint.

$$attributeConstraint ::= aec\langle\!\langle attributeExpressionConstraint \rangle\!\rangle \mid \\ \quad acvc\langle\!\langle attributeConcreteValueConstraint \rangle\!\rangle$$

`attributeExpressionEonstraint` is a combination of a source constraint (target if reverse flag is true) and an expression constraint value whose interpretation yields a set of SCTID's that filter the target (source if reverse flag).

$$[reverseFlag]$$

$$\begin{array}{|l}
\hline
\textit{attributeExpressionConstraint} \\
a : \textit{attributeSubject} \\
rf : \textit{reverseFlag}[0\,..\,1] \\
cs : \textit{expressionConstraintValue} \\
\hline
\end{array}$$

A concrete value constraint is the combination of a source constraint and a comparison operator/concrete value whose interpretation yields a set of source ids. The intersection of the source and concrete value interpretation is the interpretation of `attributeConcreteValueConstraint`

$$\textit{comparisonOperator} ::= \textit{eq} \mid \textit{neq} \mid \textit{gt} \mid \textit{ge} \mid \textit{lt} \mid \textit{le}$$

$$\begin{array}{|l}
\hline
\textit{attributeConcreteValueConstraint} \\
a : \textit{attributeSubject} \\
op : \textit{comparisonOperator} \\
v : \textit{concreteValue} \\
\hline
\end{array}$$

The interpretation of an attribute is the interpretation of the cardinality applied against the underlying attribute constraint, producing a set of SCTID's or an error condition

The interpretation of an attribute constraint is the interpretation of either the attribute expression constraint or the concrete expression constraint that produces a set of Quads or an error condition.

The actual interpretations if attribute expression and concrete value are in Section 5

$$
\begin{array}{l}
i\_attribute : \\
\qquad Substrate \nrightarrow attribute \nrightarrow Sctids\_or\_Error \\
i\_attributeConstraint : \\
\qquad Substrate \nrightarrow attributeConstraint \nrightarrow Quads\_or\_Error \\
i\_attributeExpressionConstraint : \\
\qquad Substrate \nrightarrow attributeExpressionConstraint \nrightarrow Quads\_or\_Error \\
i\_attributeConcreteValueConstraint : \\
\qquad Substrate \nrightarrow attributeConcreteValueConstraint \nrightarrow Quads\_or\_Error \\
\hline
\forall ss : Substrate;\ a : attribute \bullet \\
\quad i\_attribute\ ss\ a = i\_cardinality\ a.card\ (i\_attributeConstraint\ ss\ a.attw) \\
\\
\forall ss : Substrate;\ aw : attributeConstraint \bullet i\_attributeConstraint\ ss\ aw = \\
\quad \mathbf{if}\ aec^{\sim} aw \in attributeExpressionConstraint \\
\qquad \mathbf{then}\ i\_attributeExpressionConstraint\ ss\ (aec^{\sim} aw) \\
\quad \mathbf{else}\ i\_attributeConcreteValueConstraint\ ss\ (acvc^{\sim} aw) \\
\\
\forall ss : Substrate;\ aec : attributeExpressionConstraint;\ eca : expressionConstraintArgs \mid \\
\quad eca.atts = i\_attributeSubject\ ss\ aec.a \land \\
\quad eca.rf = aec.rf \land \\
\quad eca.subjOrTarg = i\_expressionConstraintValue\ ss\ aec.cs \bullet \\
\quad i\_attributeExpressionConstraint\ ss\ aec = i\_attributeExpression\ ss\ eca \\
\\
\forall ss : Substrate;\ awc : attributeConcreteValueConstraint;\ aca : concreteConstraintArgs \mid \\
\quad aca.atts = i\_attributeSubject\ ss\ awc.a \land \\
\quad aca.op = awc.op \land \\
\quad aca.t = awc.v \bullet \\
\quad i\_attributeConcreteValueConstraint\ ss\ awc = i\_concreteAttributeConstraint\ ss\ aca
\end{array}
$$

4.8 AttributeSet and AttributeGroup

An `attributeGroup` is an `attributeSet`. An `attributeSet` consists of a sequence of one or more attribute constraints joined by `binaryOperator`s.

$$
\begin{array}{l}
binaryOperator ::= conjunction \mid disjunction \mid exclusion \\
attributeGroup == attributeSet \\
attributeSet == attribute \times \mathrm{seq}(binaryOperator \times attribute)
\end{array}
$$

$$
\begin{array}{l}
i\_attributeGroup : Substrate \nrightarrow attributeGroup \nrightarrow Sctids\_or\_Error \\
\\
i\_attributeSet : Substrate \nrightarrow attributeSet \nrightarrow Sctids\_or\_Error \\
\hline
\forall ss : Substrate;\ ag : attributeGroup \bullet i\_attributeGroup\ ss\ ag = \\
\quad i\_attributeSet\ ss\ ag \\
\\
\forall ss : Substrate;\ a : attributeSet \bullet i\_attributeSet\ ss\ a = \\
\quad idgroups\_to\_sctids\ (evalCmpndAtt\ ss\ (i\_groupedAttribute\ ss\ (first\ a))\ (second\ a))
\end{array}
$$

4.9 Compound attribute evaluation

The left-to-right evaluation of `attributeSet`. The interpretation takes a *lhs* as a function from *SCTID* to *GROUP* and a *rhs* which is sequence of operator/attribute tuples and recursively interprets the *lhs* and interpretation of the head of the *rhs*.

$$
\begin{array}{l}
\mathit{evalCmpndAtt} : \\
\qquad \mathit{Substrate} \rightarrow \mathit{IDGroups} \rightarrow \mathrm{seq}(\mathit{binaryOperator} \times \mathit{attribute}) \rightarrow \mathit{IDGroups} \\
\mathit{gintersect}, \mathit{gunion}, \mathit{gminus}, \mathit{gfirstError} : \\
\qquad \mathit{IDGroups} \rightarrow \mathit{IDGroups} \rightarrow \mathit{IDGroups} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ \mathit{lhs} : \mathit{IDGroups};\ \mathit{rhs} : \mathrm{seq}(\mathit{binaryOperator} \times \mathit{attribute}) \bullet \\
\mathit{evalCmpndAtt}\ \mathit{ss}\ \mathit{lhs}\ \mathit{rhs} = \\
\textbf{if}\ \mathit{rhs} = \langle\rangle \\
\quad \textbf{then}\ \mathit{lhs} \\
\textbf{else if}\ \mathit{first}\,(\mathit{head}\ \mathit{rhs}) = \mathit{conjunction} \\
\textbf{then}\ \mathit{evalCmpndAtt}\ \mathit{ss}\,(\mathit{gintersect}\ \mathit{lhs}(\mathit{i\_groupedAttribute}\ \mathit{ss}\,(\mathit{second}\,(\mathit{head}\ \mathit{rhs}))))\,(\mathit{tail}\ \mathit{rhs}) \\
\textbf{else if}\ \mathit{first}\,(\mathit{head}\ \mathit{rhs}) = \mathit{disjunction} \\
\textbf{then}\ \mathit{evalCmpndAtt}\ \mathit{ss}\,(\mathit{gunion}\ \mathit{lhs}(\mathit{i\_groupedAttribute}\ \mathit{ss}\,(\mathit{second}\,(\mathit{head}\ \mathit{rhs}))))\,(\mathit{tail}\ \mathit{rhs}) \\
\textbf{else}\ \mathit{evalCmpndAtt}\ \mathit{ss}\,(\mathit{gminus}\ \mathit{lhs}(\mathit{i\_groupedAttribute}\ \mathit{ss}\,(\mathit{second}\,(\mathit{head}\ \mathit{rhs}))))\,(\mathit{tail}\ \mathit{rhs}) \\
\forall\, a, b, r : \mathit{IDGroups} \mid \\
\quad r = \textbf{if}\ \mathit{gerror}^{\sim} a \in \mathit{ERROR} \lor \mathit{gerror}^{\sim} b \in \mathit{ERROR}\ \textbf{then}\ \mathit{gfirstError}\ a\ b \\
\quad \textbf{else}\ \mathit{id\_groups}\,(\mathit{id\_groups}^{\sim} a \cap \mathit{id\_groups}^{\sim} b) \bullet \\
\quad \mathit{gintersect}\ a\ b = r \\
\forall\, a, b, r : \mathit{IDGroups} \mid \\
\quad r = \textbf{if}\ \mathit{gerror}^{\sim} a \in \mathit{ERROR} \lor \mathit{gerror}^{\sim} b \in \mathit{ERROR}\ \textbf{then}\ \mathit{gfirstError}\ a\ b \\
\quad \textbf{else}\ \mathit{id\_groups}\,(\mathit{id\_groups}^{\sim} a \cup \mathit{id\_groups}^{\sim} b) \bullet \\
\quad \mathit{gunion}\ a\ b = r \\
\forall\, a, b, r : \mathit{IDGroups} \mid \\
\quad r = \textbf{if}\ \mathit{gerror}^{\sim} a \in \mathit{ERROR} \lor \mathit{gerror}^{\sim} b \in \mathit{ERROR}\ \textbf{then}\ \mathit{gfirstError}\ a\ b \\
\quad \textbf{else}\ \mathit{id\_groups}\,(\mathit{id\_groups}^{\sim} a \setminus \mathit{id\_groups}^{\sim} b) \bullet \\
\quad \mathit{gminus}\ a\ b = r
\end{array}
$$

$$
\begin{array}{l}
\mathit{i\_groupedAttribute} : \\
\qquad \mathit{Substrate} \rightarrow \mathit{attribute} \rightarrow \mathit{IDGroups} \\
\hline
\forall\, \mathit{ss} : \mathit{Substrate};\ a : \mathit{attribute} \bullet \\
\quad \mathit{i\_groupedAttribute}\ \mathit{ss}\ a = \mathit{i\_groupCardinality}\,(\mathit{i\_attributeConstraint}\ \mathit{ss}\ a.\mathit{attw})\ a.\mathit{card}
\end{array}
$$

4.10 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
- $[m_1\,..\,n_1]op[m_2\,..\,n_2]$... – 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

TODO: This assumes that q.t is always type object. It doesn't say what to do if it is concrete **TODO:** the *quads_to_idgroups* function seems to express what is described below more simply

$$
\begin{array}{l}
i\_groupCardinality : \\
\qquad Quads\_or\_Error \rightarrow cardinality[0\,..\,1] \rightarrow IDGroups \\
\hline
\forall\, quads : Quads\_or\_Error;\ oc : cardinality[0\,..\,1];\ uniqueGroups : \mathbb{P}\, GROUP; \\
\qquad quadsByGroup : GROUP \nrightarrow \mathbb{P}\, Quad \mid \\
\quad uniqueGroups = \{q : quads\_for\ quads \bullet q.g\} \land \\
\quad quadsByGroup = \{g : uniqueGroups;\ q : \mathbb{P}\, Quad \mid \\
\qquad q = \{e : quads\_for\ quads \mid e.g = g\} \bullet g \mapsto (evalCardinality\ oc\ q)\} \bullet \\
i\_groupCardinality\ quads\ oc = \\
\quad id\_groups\,\{sctid : SCTID;\ groups : \mathbb{P}\, GROUP \mid sctid \in \{q : \bigcup(\mathrm{ran}\ quadsByGroup) \bullet \\
\qquad \mathbf{if}\ quad\_direction\ quads = source\_direction\ \mathbf{then}\ q.s\ \mathbf{else}\ object^{\sim}\, q.t\} \land \\
\quad groups = \{g : \mathrm{dom}\ quadsByGroup \mid (\exists\, q : quadsByGroup\ g \bullet \\
\qquad sctid = \mathbf{if}\ quad\_direction\ quads = source\_direction\ \mathbf{then}\ q.s\ \mathbf{else}\ object^{\sim}\, q.t)\} \bullet \\
\qquad sctid \mapsto groups\}
\end{array}
$$

4.11 Cardinality

Interpretation: *cardinality* is tested against a set of quads with the following rules:

1. Errors are propagated
2. No cardinality or passing cardinality returns the sources / targets of the set of quads
3. Otherwise return an empty set

$$
\begin{array}{l}
i\_cardinality : \\
\quad cardinality[0\,..\,1] \rightarrow Quads\_or\_Error \rightarrow Sctids\_or\_Error \\
\hline
\forall\, card : cardinality[0\,..\,1];\ quads : Quads\_or\_Error \bullet \\
i\_cardinality\ card\ quads = \\
\mathbf{if}\ quads \in \operatorname{ran} error \\
\quad \mathbf{then}\ idgroups\_to\_sctids\,(quads\_to\_idgroups\ quads) \\
\mathbf{else} \\
\quad idgroups\_to\_sctids\,(quads\_to\_idgroups \\
\qquad (quad\_value\,(evalCardinality\ card\,(quads\_for\ quads), quad\_direction\ quads)))
\end{array}
$$

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 $\emptyset$

$$
\begin{array}{l}
[T] \\
evalCardinality : cardinality[0\,..\,1] \rightarrow \mathbb{P}\,T \rightarrow \mathbb{P}\,T \\
\hline
\forall\, opt\_cardinality : cardinality[0\,..\,1];\ t : \mathbb{P}\,T \bullet \\
evalCardinality\ opt\_cardinality\ t = \\
\quad \mathbf{if}\ \#opt\_cardinality = 0 \lor \\
\qquad (\#t \geq first\,(head\ opt\_cardinality)) \land \\
\qquad (second\,(head\ opt\_cardinality) = many \lor \\
\qquad num^{\sim}(second\,(head\ opt\_cardinality)) \geq \#t) \\
\quad \mathbf{then}\ t \\
\quad \mathbf{else}\ \emptyset
\end{array}
$$

5 Substrate Interpretations

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

5.1 attributeName

`attributeName` is the interpretation of a `conceptReference` with the additional caveat that the SCTID(s) have to be substrate *attributeIds*

```
attributeName = conceptReference
```

$attributeName == conceptReference$

$$
\begin{array}{l}
i\_attributeName : \\
\quad Substrate \rightarrow attributeName \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\ attName : attributeName \bullet \\
(\mathbf{let}\ attn == i\_conceptReference\ ss\ attName \bullet \\
\quad \mathbf{if}\ attn \in \operatorname{ran} ERROR \\
\quad\quad \mathbf{then}\ attn \\
\quad \mathbf{else\ if}\ (result\_sctids\ attn) \subseteq ss.attributeIds \\
\quad\quad \mathbf{then}\ attn \\
\quad \mathbf{else}\ error\ unknownAttributeId)
\end{array}
$$

5.2 attributeExpressionConstraint

`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.

$$
\begin{array}{l}
i\_attributeExpressionConstraint : \\
\quad Substrate \rightarrow reverseFlag[0 \mathrel{..} 1] \rightarrow Sctids\_or\_Error \rightarrow \\
\qquad expressionComparisonOperator \rightarrow Sctids\_or\_Error \rightarrow Quads\_or\_Error \\
\hline
\forall ss : Substrate;\ rf : reverseFlag[0 \mathrel{..} 1];\ atts : Sctids\_or\_Error; \\
\quad op : expressionComparisonOperator;\ subj\_or\_targets : Sctids\_or\_Error \bullet \\
i\_attributeExpressionConstraint\ ss\ rf\ atts\ op\ subj\_or\_targets = \\
\mathbf{if}\ atts \in \operatorname{ran} error \lor subj\_or\_targets \in \operatorname{ran} error \\
\quad \mathbf{then}\ qfirstError\{atts, subj\_or\_targets\} \\
\mathbf{else\ if}\ \#args.rf = 0 \land op = eco\_eq\ \mathbf{then} \\
\quad quad\_value(\{t : result\_sctids\ subj\_or\_targets;\ a : result\_sctids\ atts; \\
\qquad rels : ss.relationships\ | \\
\qquad object^{\sim}\ rels.t = t \land rels.a = a \bullet rels\}, source\_direction) \\
\mathbf{else\ if}\ \#args.rf = 1 \land op = eco\_eq\ \mathbf{then} \\
\quad quad\_value(\{s : result\_sctids\ subj\_or\_targets;\ a : result\_sctids\ atts; \\
\qquad rels : ss.relationships\ | \\
\qquad rels.s = s \land rels.t \in \operatorname{ran} object \land rels.a = a \bullet rels\}, targets\_direction) \\
\mathbf{else\ if}\ \#args.rf = 0 \land op = eco\_neq\ \mathbf{then} \\
\quad quad\_value(\{t : result\_sctids\ subj\_or\_targets;\ a : result\_sctids\ atts; \\
\qquad rels : ss.relationships\ | \\
\qquad object^{\sim}\ rels.t \neq t \land rels.a = a \bullet rels\}, source\_direction) \\
\mathbf{else\ if}\ \#args.rf = 1 \land op = eco\_neq\ \mathbf{then} \\
\quad quad\_value(\{s : result\_sctids\ subj\_or\_targets;\ a : result\_sctids\ atts; \\
\qquad rels : ss.relationships\ | \\
\qquad rels.s \neq s \land rels.t \in \operatorname{ran} object \land rels.a = a \bullet rels\}, targets\_direction)
\end{array}
$$

5.3 concreteAttributeConstraint

$$
\begin{array}{l}
i\_concreteAttributeConstraint : \\
\quad Substrate \rightarrow Sctids\_or\_Error \rightarrow concreteComparisonOperator \rightarrow \\
\qquad concreteValue \rightarrow Quads\_or\_Error \\
\hline
\forall ss : Substrate;\ atts : Sctids\_or\_error;\ op : concreteComparisonOperator; \\
\qquad val : concreteValue \bullet \\
i\_concreteAttributeConstraint = \\
\mathbf{if}\ atts \in \operatorname{ran} error \\
\quad \mathbf{then}\ qerror\ (error^{\sim}\ atts) \\
\mathbf{else}\ quad\_value\{ss.relationships \mid ss.a \in (result\_sctids\ atts) \land \\
\qquad ss.t \in \operatorname{ran} concrete \land val \in concreteMatch\ (concrete^{\sim}\ ss.t)\ op\}
\end{array}
$$

$$
\begin{array}{l}
concreteMatch : \\
\quad CONCRETEVALUE \rightarrow comparison \rightarrow concreteValue
\end{array}
$$

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.

$$
\begin{array}{l}
i\_constraintOperator : \\
\quad Substrate \rightarrow constraintOperator[0..1] \rightarrow Sctids\_or\_Error \rightarrow Sctids\_or\_Error \\
completeFun : (SCTID \rightarrow \mathbb{P}\,SCTID) \rightarrow SCTID \rightarrow \mathbb{P}\,SCTID \\
\hline
\forall\, ss : Substrate;\ oco : constraintOperator[0..1];\ subresult : Sctids\_or\_Error \bullet \\
i\_constraintOperator\ ss\ oco\ subresult = \\
\quad \textbf{if}\ error^{\sim} subresult \in ERROR \lor \#oco = 0 \\
\qquad \textbf{then}\ subresult \\
\quad \textbf{else if}\ head\ oco = descendantOrSelfOf \\
\qquad \textbf{then}\ ok(\bigcup\{id : result\_sctids\ subresult \bullet \\
\qquad\qquad completeFun\ ss.descendants\ id\} \cup result\_sctids\ subresult) \\
\quad \textbf{else if}\ head\ oco = descendantOf \\
\qquad \textbf{then}\ ok(\bigcup\{id : result\_sctids\ subresult \bullet \\
\qquad\qquad completeFun\ ss.descendants\ id\,\}) \\
\quad \textbf{else if}\ head\ oco = ancestorOrSelfOf \\
\qquad \textbf{then}\ ok(\bigcup\{id : result\_sctids\ subresult \bullet \\
\qquad\qquad completeFun\ ss.ancestors\ id\} \cup result\_sctids\ subresult) \\
\quad \textbf{else}\ ok(\bigcup\{id : result\_sctids\ subresult \\
\qquad\qquad \bullet\ completeFun\ ss.ancestors\ id\,\}) \\
\forall\, f : (SCTID \rightarrow \mathbb{P}\,SCTID);\ id : SCTID \bullet completeFun\, f\ id = \\
\quad \textbf{if}\ id \in \mathrm{dom}\, f\ \textbf{then}\ f\ id\ \textbf{else}\ \emptyset
\end{array}
$$

5.4 FocusConcept

`focusConcept = [memberOf] conceptReference`

5.4.1 focusConcept

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

$$
\begin{array}{l}
focusConcept ::= \\
\quad focusConcept\_m\langle\langle conceptReference\rangle\rangle\ | \\
\quad focusConcept\_c\langle\langle conceptReference\rangle\rangle
\end{array}
$$

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

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

5.4.2 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*

$$
\begin{array}{l}
i\_memberOf : Substrate \rightarrow Sctids\_or\_Error \rightarrow Sctids\_or\_Error \\
i\_refset : Substrate \rightarrow SCTID \rightarrow Sctids\_or\_Error \\
\hline
\forall\, ss : Substrate;\; refsetids : Sctids\_or\_Error \bullet \\
i\_memberOf\; ss\; refsetids = \\
\quad \textbf{if } refsetids \in \operatorname{ran} error \\
\quad\quad \textbf{then } refsetids \\
\quad \textbf{else } bigunion\{sctid : result\_sctids\; refsetids \bullet i\_refset\; ss\; sctid\} \\
\\
\forall\, ss : Substrate;\; sctid : SCTID \bullet \\
i\_refset\; ss\; sctid = \\
\textbf{if } sctid \notin ss.refsetIds \\
\quad \textbf{then } error\; unknownRefsetId \\
\textbf{else if } sctid \in \operatorname{dom} ss.refsetIds \\
\quad \textbf{then } ok\; (ss.refsets\; sctid) \\
\textbf{else } ok\; \emptyset
\end{array}
$$

5.5 ConceptReferences

5.5.1 conceptId

```
conceptId = sctId
```

Interpretation: `conceptId` is interpreted as *SCTID* that it represents. For our purposes, all `conceptId`s are considered valid, so this is a bijection.

$[conceptId]$

$$
i\_conceptId : conceptId \rightarrowtail\!\!\!\!\twoheadrightarrow SCTID
$$

5.5.2 conceptReference

```
conceptReference = conceptId [ "|" Term "|"]
conceptId = sctId
```

Interpretation: `conceptReference` is interpreted as the *set of SCTIDs* that are equivalent to the supplied SCTID if it is known concepts, c, in the substrate. If it isn't in the list of known concepts otherwise as the *unknownConceptReference* error. The `Term` part of `conceptReference` is ignored.

$$conceptReference == conceptId$$

$$i\_conceptReference : Substrate \rightarrow conceptReference \rightarrow Sctids\_or\_Error$$

$$\begin{array}{l} \forall\, ss : Substrate;\ c : conceptReference \bullet i\_conceptReference\ ss\ c = \\ \quad (\mathbf{let}\ sctid == i\_conceptId\ c \bullet \\ \quad \mathbf{if}\ sctid \in ss.concepts\ \mathbf{then}\ ok\,(ss.equivalent\_concepts\ sctid) \\ \quad \mathbf{else}\ error\ unknownConceptReference) \end{array}$$

6 Glue and Helper Functions

This section carries various type transformations and error checking functions

6.1 Types

- **direction** – an indicator whether a collection of quads was determined as source to target (*source_direction*) or target to source (*targets_direction*)
- **Quads_or_Error** – a collection of *Quad*s or an error condition. If it is a collection *Quad*s, it also carries a direction indicator that determines whether it represents a set of sources or targets.
- **IDGroups** – a map from *SCTID*s to the *GROUP* they were in when they passed if successful, otherwise an error indication.

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

$$IDGroups ::= id\_groups\langle\langle SCTID \pfun \mathbb{P}\,GROUP\rangle\rangle \mid gerror\langle\langle ERROR\rangle\rangle$$

6.2 Result transformations

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

- **to_idGroups** – the *SCTID* to *GROUP* 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 sctids in an idgroup sans the group identifiers.

$$
\begin{array}{l}
result\_sctids : Sctids\_or\_Error \rightarrow \mathbb{P}\, SCTID \\
quads\_for : Quads\_or\_Error \rightarrow \mathbb{P}\, Quad \\
quad\_direction : Quads\_or\_Error \nrightarrow direction \\
to\_idGroups : IDGroups \rightarrow SCTID \rightarrow \mathbb{P}\, GROUP \\
quads\_to\_idgroups : Quads\_or\_Error \rightarrow IDGroups \\
idgroups\_to\_sctids : IDGroups \rightarrow \mathbb{P}\, SCTID \\
\hline
\forall r : Sctids\_or\_Error \bullet result\_sctids\ r = \\
\quad \mathbf{if}\ error^\sim r \in ERROR\ \mathbf{then}\ \emptyset \\
\quad \mathbf{else}\ ok^\sim r \\[1ex]
\forall q : Quads\_or\_Error \bullet quads\_for\ q = \\
\quad \mathbf{if}\ qerror^\sim q \in ERROR\ \mathbf{then}\ \emptyset \\
\quad \mathbf{else}\ first\,(quad\_value^\sim q) \\[1ex]
\forall q : Quads\_or\_Error \bullet quad\_direction\ q = \\
\quad second\,(quad\_value^\sim q) \\[1ex]
\forall g : IDGroups \bullet to\_idGroups\ g = \\
\quad \mathbf{if}\ gerror^\sim g \in ERROR\ \mathbf{then}\ \emptyset \\
\quad \mathbf{else}\ id\_groups^\sim g \\[1ex]
\forall q : Quads\_or\_Error \bullet quads\_to\_idgroups\ q = \\
\quad \mathbf{if}\ qerror^\sim q \in ERROR\ \mathbf{then}\ gerror(qerror^\sim q) \\
\quad \mathbf{else\ if}\ quad\_direction\ q = source\_direction \\
\qquad \mathbf{then}\ id\_groups\,\{s : SCTID \mid (\exists\, qr : quads\_for\ q \bullet s = qr.s) \bullet \\
\qquad s \mapsto \{qr : quads\_for\ q \bullet qr.g\}\} \\
\quad \mathbf{else} \\
\qquad id\_groups\,\{t : SCTID \mid (\exists\, qr : quads\_for\ q \bullet t = object^\sim qr.t) \bullet \\
\qquad t \mapsto \{qr : quads\_for\ q \bullet qr.g\}\} \\[1ex]
\forall g : IDGroups \bullet idgroups\_to\_sctids\ g = \\
\quad \mathbf{if}\ gerror^\sim g \in ERROR\ \mathbf{then}\ \emptyset \\
\quad \mathbf{else}\ \mathrm{dom}(id\_groups^\sim g)
\end{array}
$$

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

- **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.

$$
\begin{array}{l}
\mathit{firstError} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \pfun \mathit{Sctids\_or\_Error} \\
\mathit{qfirstError} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \pfun \mathit{Quads\_or\_Error} \\[6pt]
\mathit{union}, \mathit{intersect}, \mathit{minus} : \mathit{Sctids\_or\_Error} \rightarrow \mathit{Sctids\_or\_Error} \rightarrow \\
\qquad\qquad \mathit{Sctids\_or\_Error} \\
\mathit{bigunion}, \mathit{bigintersect} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \rightarrow \mathit{Sctids\_or\_Error}
\end{array}
$$

$$
\begin{array}{l}
\forall\, x, y : \mathit{Sctids\_or\_Error} \bullet \mathit{union}\; x\; y = \\
\quad \mathbf{if}\; x \in \operatorname{ran} \mathit{error} \lor y \in \operatorname{ran} \mathit{error}\; \mathbf{then}\; \mathit{firstError}\, \{x, y\} \\
\quad \mathbf{else}\; \mathit{ok}\, ((\mathit{ok}^{\sim} x) \cup (\mathit{ok}^{\sim} y)) \\[6pt]
\forall\, x, y : \mathit{Sctids\_or\_Error} \bullet \mathit{intersect}\; x\; y = \\
\quad \mathbf{if}\; x \in \operatorname{ran} \mathit{error} \lor y \in \operatorname{ran} \mathit{error}\; \mathbf{then}\; \mathit{firstError}\, \{x, y\} \\
\quad \mathbf{else}\; \mathit{ok}\, ((\mathit{ok}^{\sim} x) \cap (\mathit{ok}^{\sim} y)) \\[6pt]
\forall\, x, y : \mathit{Sctids\_or\_Error} \bullet \mathit{minus}\; x\; y = \\
\quad \mathbf{if}\; x \in \operatorname{ran} \mathit{error} \lor y \in \operatorname{ran} \mathit{error}\; \mathbf{then}\; \mathit{firstError}\, \{x, y\} \\
\quad \mathbf{else}\; \mathit{ok}\, ((\mathit{ok}^{\sim} x) \setminus (\mathit{ok}^{\sim} y)) \\[6pt]
\forall\, \mathit{rs} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \bullet \mathit{bigunion}\; \mathit{rs} = \\
\quad \mathbf{if}\; \exists\, r : \mathit{rs} \bullet r \in \operatorname{ran} \mathit{error}\; \mathbf{then}\; \mathit{firstError}\; \mathit{rs} \\
\quad \mathbf{else}\; \mathit{ok}\, (\bigcup \{ r : \mathit{rs} \bullet \mathit{result\_sctids}\; r \}) \\[6pt]
\forall\, \mathit{rs} : \mathbb{P}\, \mathit{Sctids\_or\_Error} \bullet \mathit{bigintersect}\; \mathit{rs} = \\
\quad \mathbf{if}\; \exists\, r : \mathit{rs} \bullet r \in \operatorname{ran} \mathit{error}\; \mathbf{then}\; \mathit{firstError}\; \mathit{rs} \\
\quad \mathbf{else}\; \mathit{ok}\, (\bigcap \{ r : \mathit{rs} \bullet \mathit{result\_sctids}\; r \})
\end{array}
$$

7 Appendix 1

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

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