SNOMED Clinical Terms®
Transforming Expressions to Normal Forms

EXTERNAL DRAFT FOR COMMENT

Version 55 (31-Jan-07)

STATUS

This document contains material useful to technical implementers working to incorporate SNOMED CT into software. However, this current document is a revision of an earlier document, which was issued by the SNOMED International Editorial Board (SIEB) as an External Draft. This revision has been issued for review prior to wider release. It is therefore subject to revision and correction without notice.

This document is intended to assist rather than direct or constrain developers. Implementers who make development decisions based on the advice herein do so entirely at their own risk.

The International Health Terminology Standards Development Organisation and the authors have applied their best endeavors to this document but do not offer any warranty in respect of the accuracy or appropriateness of the information for implementation in any technical environment.

Table of Contents

Document History

Version	Date	Notes
1	04-Apr-05	Initial draft for discussion by Concept Model Design Team
2	31-May-05	Complete re-write based on comments at CMDT and results of testing for discussion by Concept Model Working Group
3	09-Jul-05	Corrections and minor revisions agreed by CMWG and approved by TSG and SIEB for release as an "External Draft"
4	20-Jan-06	Revision based on comment period input and further experience. Addition of new sections containing examples, Clarification of rules for merging groups and attributes. Detailed description of the process of testing expression subsumption. A new section on the use of an "expressions repository" and associated optimizations
5	31-Aug-06	Revision based on agreement to add advice on handling the finding context of "known absent" based on white paper as approved by the CMWG meeting in June 2006. o Most of the text of that paper included as an Annex o Addition of absent finding testing specific subsumption testing rules Also clarified subsumption testing rules by rewording and restructuring while integrating the exception handling for absent finding. Various corrections in response to comments.
5a	31-Jul-07	Document updated to reflect the transfer of SNOMED CT to the International Health Terminology Standards Development Organisation (IHTSDO)
5b	31-Jan-08	Copyright notices updated to 2008

1 Status

This guide to generating normal forms was developed by the SNOMED Concept Model Working Group and has been tested using a demonstration version of the CliniClue™ Browser. The previous revision of this document was approved by the SNOMED International Editorial Board meeting in June 2005 for release as an External Draft Guidance Document. This revision takes account of comments received but remains subject to revision and comments contributing to correction or clarification of the guide are welcomed.

Many of the ideas in this document are introduced and explained in the SNOMED CT draft document on "Abstract Logical Models and Representational forms". That document is an essential source of reference for readers of this document and should be read first by those who are unfamiliar with the subject.

2 Introduction

The purpose of generating normal forms is to facilitate complete and accurate retrieval of pre and post-coordinated SNOMED CT expressions from clinical records or other resources.

The approach described is based on the description logic definitions of SNOMED CT concepts that are used when recording clinical statements in an electronic records system. Using this approach, expressions that are authored, stored and/or communicated in a relatively informal close-to-user form are logically transformed into a common normalized form. In this normalized form it is possible to apply simple rules to test subsumption between expressions.

The simplest case of a valid close-to-user expression is a single conceptId, and the approach described can be applied to these simple pre-coordinated expressions, as well as to more complex expressions that include multiple conceptIds and refinements (qualifiers).

The approach to normalization may be applied to specific SNOMED CT expressions but may also be extended to take account of contextual information derived from the information model in which the expression is situated. Therefore, the normal form may include SNOMED CT context information, even if this is not present in the initial SNOMED CT expression.

The algorithm extends earlier work on canonical forms as follow:

- Normalizes fully-defined values within definitions or expressions producing nested expressions that are fully normalized.
- Merges refinements stated in an expression with definitional relationships present in the definitions of the concepts referenced by the expression.
  - The merge process takes account of refinements that may not be grouped or nested in a manner that precisely reflects the structure of a current (or future) concept definition.
  - This avoids the need to add, store and communicate potentially spurious detail from current definitions to the expression recorded by a user or software application.
- Takes account of context rules including soft default context and a preliminary approach to moodCode mapping and handling of procedures with values (present in algorithm but not yet easily visible in test environment).
- Supports subsumption tests that take account of finding specified with “known absent” finding context.

3 Expressions

3.1 Candidate and predicate expressions

This paper considers transformations that can be applied to expressions, to enable effective subsumption testing. In any subsumption test there are two expressions, one of which is being tested for subsumption by the other. To distinguish these expressions the following definitions are used:

Candidate expression – An expression that is being tested to see if it is subsumed by another expression.

Predicate expression – An expression that is being tested to see if it subsumes another expression.

Examples

Predicate	Candidate	Test result
Fracture of femur	**Fracture of neck of femur**	**True**
	Fracture of bone	**False**
Fracture of bone	**Fracture of femur**	**True**
	Fracture of neck of femur	**True**
Asthma ***(in patient)***	**Family history of asthma**	**False**
	Severe asthma ***(in patient)***	**True**
Family history of respiratory disease	**Family history of asthma**	**True**

3.2 Expression parts

The figures in this section illustrate some terms that are used to describe different parts of an expression throughout the remainder of this document. This is intended for reference within this document only.

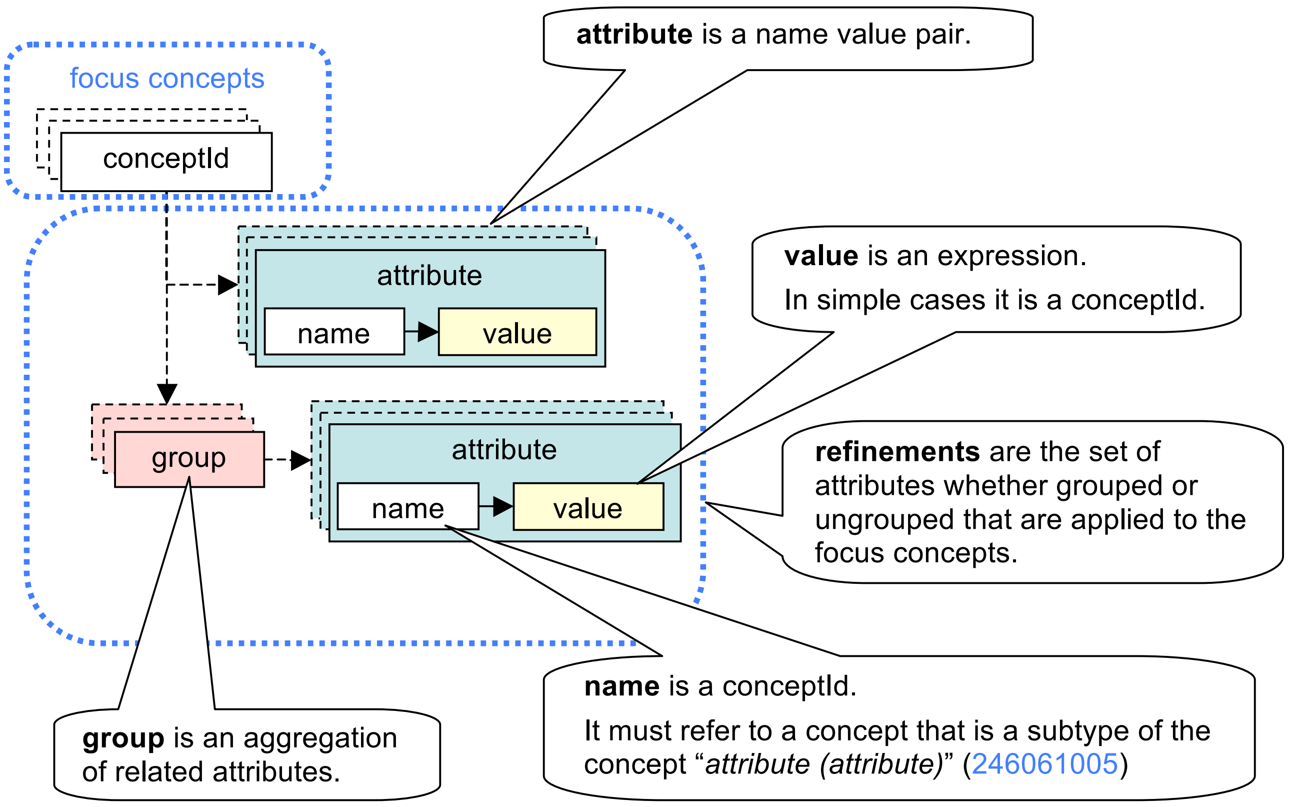

attribute is a name value pair.
focus concepts
conceptId
attribute
name
value
value is an expression.
In simple cases it is a conceptId.
group
attribute
name
value
refinements are the set of attributes whether grouped or ungrouped that are applied to the focus concepts.
group is an aggregation of related attributes.
name is a conceptId.
It must refer to a concept that is a subtype of the concept "*attribute (attribute)*" (246061005)

Figure 1. Illustration of names used to refer to general elements of an expression

As illustrated by Figure 1, an expression consists of one or more conceptIds plus optional refinements. The refinements may include any number of attributes. Attributes are expressed as name-value pairs and may apply independently or as part of a group.

The name part of the attribute name-value pair is a conceptId that refers to a concept that names the characteristic that is refined by this attribute. The value part of the attribute name-value pair is an expression. In simple cases, this is simply a conceptId referring to a concept that represents the appropriate value for this attribute. However, it may also be a nested expression as shown in Figure 2.

Figure 2 illustrates the potential for nesting of expressions and the naming conventions applied in this document to distinguish different parts of an expression at different levels. The top level of an expression is referred to as the "focus expression". It consists of a set of one or more "focus concepts" and a "focus refinement". The values of the attributes in the focus refinement are "nested expressions" that consist of one or more "value concepts" optionally refined by a "nested refinement".

Expressions may be nested recursively so there may be further levels of "nested expressions" with "nested refinements". If it is necessary to distinguish the level of nesting, the following naming convention is applied

Level number	Description
level 0 expression	**Focus expression**
level 1 expression	**Nested expression**
level *N* expression	**An expression nested inside a level (*N* - 1) expression**

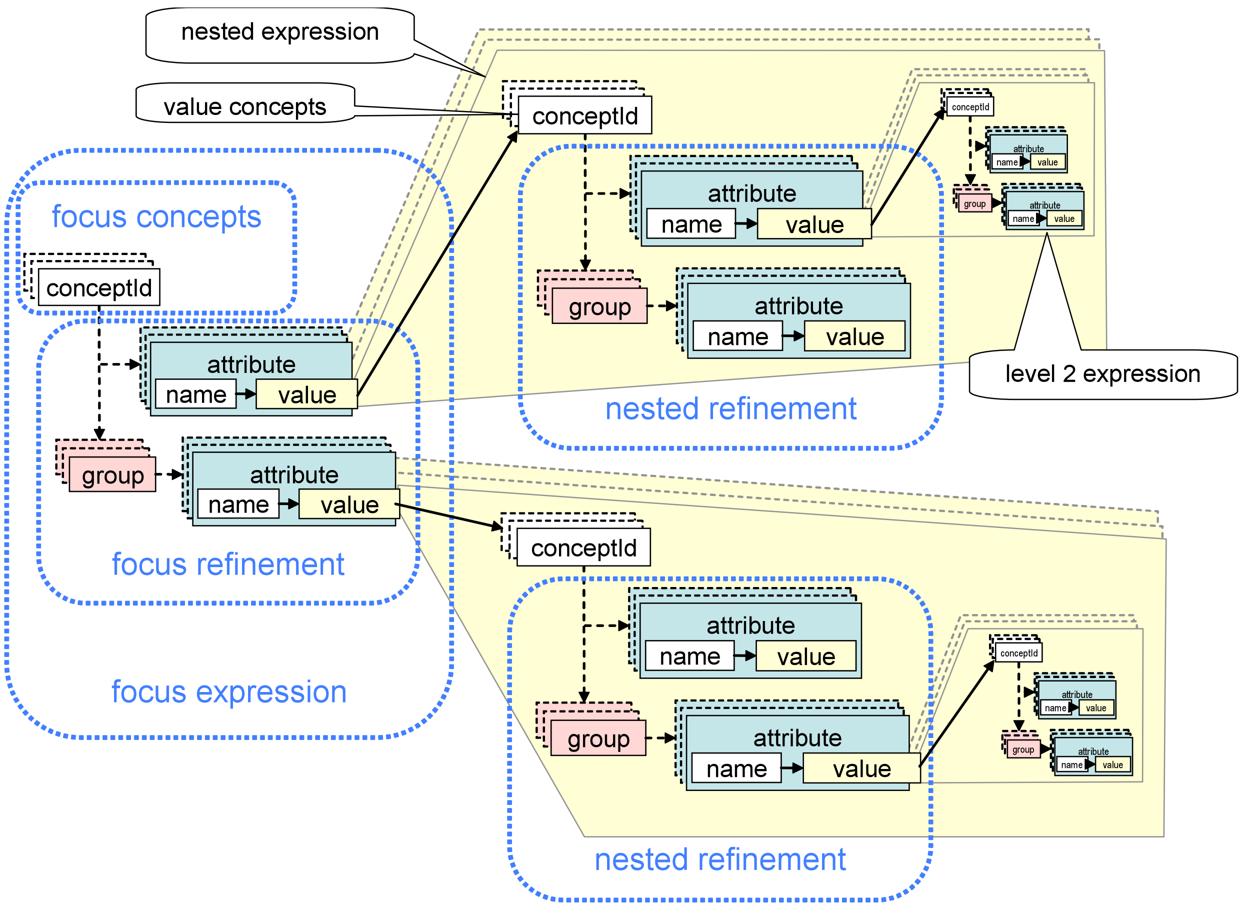


Figure 2. Illustration of the names used to refer to parts of a nested expression

The general pattern shown in Figure 2 applies to all expressions whether or not they include SNOMED CT context information. Figure 3 illustrates the specific features of an expression that includes a representation of SNOMED CT context.

The "focus expression" of a context containing expression is the "context wrapper" and may include a "context refinement" consisting of a set of context attributes:

- *associated finding* or *associated procedure*
- *finding context* or *procedure context*
- *subject relationship context*
- *temporal context*

In a normalized context expression, all context attributes are grouped. Each group in a normalized context wrapper contains a complete set of four context attributes[1].

The value of the *associated finding* or *associated procedure* is a "nested expression" which is referred to as the "clinical kernel".

During some stages of processing, the "clinical kernel" is separated from the "context wrapper". When separated from its context the "clinical kernel" is the "focus expression" of a context-free expression.

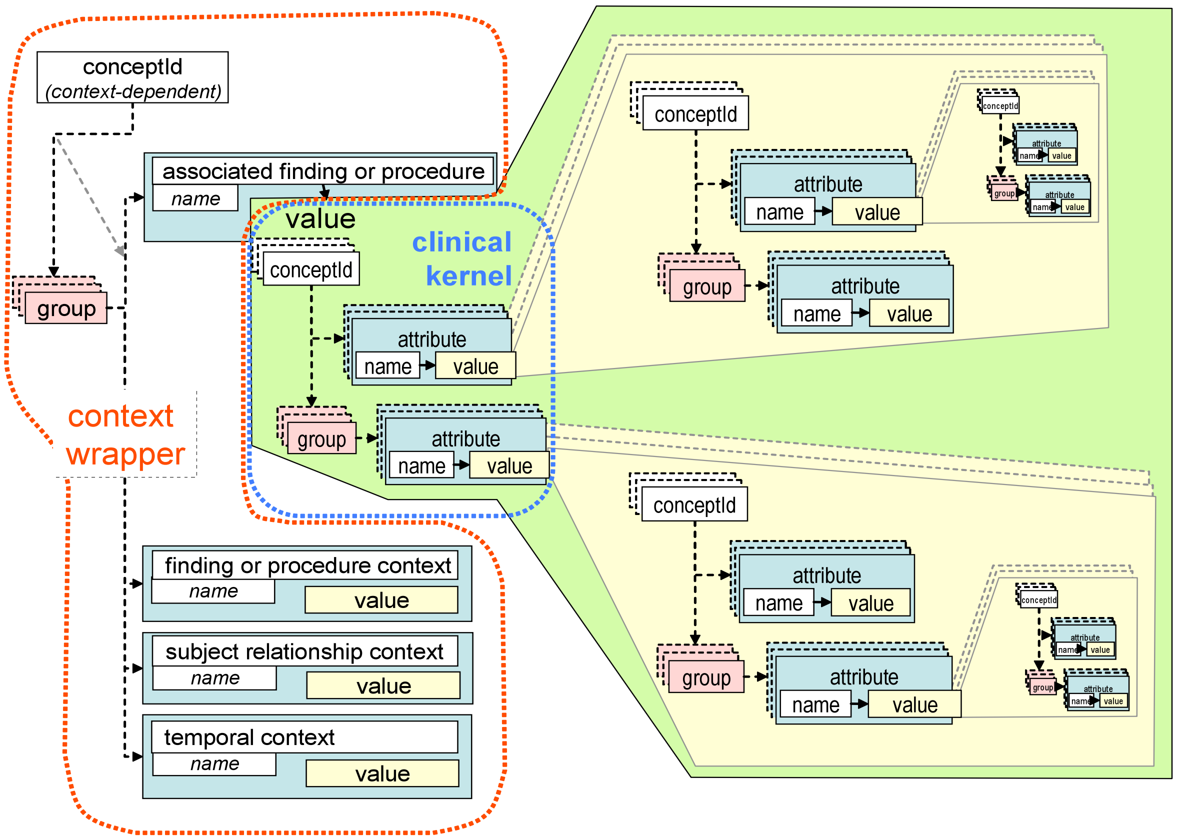


Figure 3. Illustration of the names used to refer to parts of an expression that represent context

[1] Usually a single group is present in a context expression. Theoretical cases exist for multiple groups where different contexts apply to different aspects of a concept but these cases are beyond the scope of the normalization rules in this document.

4 Normal forms

4.1 Introduction

A normal form is a view that can be generated for any valid expression by applying a set of logical transformation rules. Once converted to their normal forms, expressions can be more easily tested for subsumption by one another.

4.2 General characteristics of normal forms

All the conceptIds present in a normal form expression refer to primitive concepts. When normalizing an expression, every conceptId is replaced with the normal form expression that represents the definition of the referenced concept.

Normalization is recursive so that any element of a concept definition that refers to another fully defined concept is also replaced by the normal form of that concept.

One test of normalization is that applying the rules to an already normalized expression should return an identical expression.

4.3 Rationale for long and short normal forms

There are two distinct normal forms that are of value when computing subsumption.

The long normal form is appropriate for a candidate expression because it explicitly states all the attributes can be inferred from concepts referenced by the expression. This makes it easier to test whether the candidate fulfils a set of predicate conditions.

The short normal form is more appropriate for predicate expressions. It enables more efficient retrieval testing because there are fewer conditions to test. However, there is no loss of specificity because any candidate that fulfils the conditions of the short normal form inevitably fulfils the conditions of the long normal form.

4.4 Building long and short normal forms

The most effective approach to building either normal form is to start by generating the long normal form. If the short normal form is required this can then be derived by removing redundant defining relationships.

Generating a long form to derive short form may appear counterintuitive. However, there are three reasons why this approach is strongly recommended.

- The process of generating either normal form includes steps that test subsumption between different parts of an expression. The long normal form is required as the predicate for these tests.
- A single approach requires only one algorithm to be specified and implemented. This eases maintenance and reduces the risks of inconsistencies developing between the two transforms.
- The short form is needed less frequently than the long form because it is used in predicates (e.g. queries) rather than in candidate instances (e.g. expression in a record). An approach that optimizes long form generation is therefore advantageous.

The next section sets out the general approach to generation of the long and short normal forms for a concept definition. References to individual concepts in the source expression are replaced by normal form concept definitions when generating the normal form of an expression.

4.5 Concept definitions in normal forms

4.5.1 Long Normal Form

A form which when applied to a candidate expression allows effective computation of whether it is subsumed by a predicate expression.

Supertype view: **Proximal Primitive Supertypes**

- For fully-defined concepts compute the proximal primitives
- For primitive concepts treat the concept itself as the proximal primitive supertype.
  - Rationale: This primitive concept must be present to enable the candidate expression to be subsumed by a predicate expression that includes this particular primitive concept.

Attribute view: **All Defining Relationships**

- For all concepts (whether fully-defined or primitive) include all non-subtype defining relationships, irrespective of whether these are also present in the union of the definitions of the primitive supertypes.
  - Rationale: An expression may be subsumed by a concept that does not share all its proximal primitive supertypes. Some of the characteristics specified as part of other primitives in the candidate expression may also be present in the candidate expression.

4.5.2 Short Normal Form

A form which when applied to a predicate expression allows effective computation of whether a candidate expression is one of its subtypes.

Supertype view: **Proximal Primitive Supertypes**

- For fully-defined concepts compute the proximal primitives
- For primitive concepts treat the concept itself as the proximal primitive supertype.
  - Rationale: As for long form see 4.5.1.

Attribute view: **Differential Defining Relationships** (compared to supertype view)

- For primitive concepts there are no differential defining relationships because the primitive concept is its own proximal primitive supertype. Therefore in predicate normal form the attribute view is empty for primitive concepts.
- For fully-defined concepts the differential form only includes defining relationships, and relationship groups, that are more specific than those present in the union of the definitions of the primitive supertypes.
  - Rationale: Each element in the predicate specifies an additional test to be applied to candidate expressions. However these additional tests are superfluous because:
    - The candidate expression cannot be subsumed by the predicate unless every candidate primitive supertype is subsumed by at least one predicate primitive supertype.
    - If this condition is met, then all defining relationships or relationship groups or the candidate primitive supertypes are inevitably also shared by the candidate expression.

4.5.3 Examples of normal form concept definitions

4.5.3.1 Normal form of a fully-defined concept with no intermediate primitives

The concept "fracture of femur" (Figure 4) is fully defined and its proximal primitive supertype is a high-level primitive[2]. This proximal primitive does not share any of the defining relationships of the concept itself. Therefore, the long and short normal forms of "fracture of femur" are identical because all its defining relationships differ from those of its primitive supertype.

Figure 4. Normal forms of a fully-defined concept with no intermediate primitive supertypes

Concept	71620000 | fracture of femur |
Definition *(distributed)*	116680003 | is a | = 64572001 | disease | {116676008 | associated morphology | = 72704001 | fracture | ,363698007 | finding site | = 71341001 | bone structure of femur | }
Long NF *(candidate)*	64572001 | disease | : {116676008 | associated morphology | = 72704001 | fracture | ,363698007 | finding site | = 71341001 | bone structure of femur | }
Short NF *(predicate)*	64572001 | disease | : {116676008 | associated morphology | = 72704001 | fracture | ,363698007 | finding site | = 71341001 | bone structure of femur | }

Using the long normal form

To test if "fracture of femur" is subsumed by another expression the long normal form is used as the candidate. If all the conditions of a predicate expression are satisfied by this candidate then "fracture of femur" is subsumed by this predicate.

- The concept "fracture of femur" is subsumed by any normal form predicate expression with a focus concept "disease" (or a supertype of "diseases" such as "clinical finding") unless the predicate expression also has conditions that do not subsume "morphology"="fracture" and "finding site" = "bone structure of femur".

Using the short normal form

To test if "fracture of femur" subsumes another expression the short normal form is used as the predicate. Any candidate expression that satisfies all the conditions of this candidate is subsumed by "fracture of femur"

- The concept "fracture of femur" subsumes any concept that is a "disease" with a morphology subsumed by "fracture" and a "finding site" subsumed by "bone structure of femur".
    - The candidate expression "disease" with "morphology"="fracture, open" and "finding site" = "structure of neck of femur" is thus subsumed.

[2] A high-level primitive is a concept that is primitive and has no fully-defined supertypes.

4.5.3.2 Normal form of a primitive concept

The concept "asthma" (Figure 5) is primitive so it is its own proximal primitive supertype. The long normal form therefore consists of the concept itself and all its defining relationships. The short normal form is simply the concept itself.

Figure 5. Normal forms of a primitive concept

Concept	195967001 | asthma | [Primitive]
Definition *(distributed)*	116680003 | is a | = 41427001 | disorder of bronchus | ,116680003 | is a | = 79688008 | respiratory obstruction | {116676008 | associated morphology | = 26036001 | obstruction | ,363698007 | finding site | = 955009 | bronchial structure | }
Long NF *(candidate)*	195967001 | asthma | : {116676008 | associated morphology | = 26036001 | obstruction | ,363698007 | finding site | = 955009 | bronchial structure | }
Short NF *(predicate)*	195967001 | asthma |

Using the long normal form

To test if "asthma" is subsumed by another expression the long normal form is used as the candidate. If all the conditions of a predicate expression are satisfied by this candidate then "asthma" is subsumed by this predicate.

- The concept "asthma" is subsumed by any normal form predicate expression with a focus concept that is "asthma" (or a supertype of "asthma" such as "disease" or "clinical finding") unless the predicate expression also has conditions that do not subsume "morphology"="obstruction" and "finding site" = "bronchial structure".

Using the short normal form

To test if "asthma" subsumes another expression the short normal form is used as the predicate. Any candidate expression that satisfies all the conditions of this candidate is subsumed by "asthma".

- The concept "asthma", only subsumes expressions that explicitly include a focus concept that is either "asthma" or a subtype of "asthma".
    - The candidate expression "disease" with "morphology" = "obstruction" and "finding site" = "bronchial obstruction" is <u>not</u> subsumed by "asthma".

4.5.3.3 *Normal form of a fully-defined concept with an intermediate primitive*

The concept "allergic asthma" (Figure 6) is fully-defined but its proximal primitive supertype ("asthma") is an intermediate primitive[3]. The long normal form consists of the proximal primitive supertype and all the defining relationships of "allergic asthma". The short normal form is the same proximal primitive but the only relationship included is "due to"="allergic reaction" as this is its only difference from the definition of the primitive.

Figure 6. Normal forms of a fully-defined concept with an intermediate primitive supertype

Concept	389145006 | allergic asthma |
Definition *(distributed)*	116680003 | is a | = 195967001 | asthma | ,116680003 | is a | = 418168000 | disorder due to allergic reaction | ,42752001 | due to | = 419076005 | allergic reaction | {116676008 | associated morphology | = 26036001 | obstruction | ,363698007 | finding site | = 955009 | bronchial structure | }
Long NF *(candidate)*	195967001 | asthma | : 42752001 | due to | = 419076005 | allergic reaction | {116676008 | associated morphology | = 26036001 | obstruction | ,363698007 | finding site | = 955009 | bronchial structure | }
Short NF *(predicate)*	195967001 | asthma | : 42752001 | due to | = 419076005 | allergic reaction |

Using the long normal form

To test if "allergic asthma" is subsumed by another expression the long normal form is used as the candidate. If all the conditions of a predicate expression are satisfied by this candidate then "allergic asthma" is subsumed by this predicate.

- The concept "allergic asthma" is subsumed by any normal form predicate expression with a focus concept that is "asthma" (or a supertype of "asthma" such as "disease" or "clinical finding") unless the predicate expression also has conditions that do not subsume "morphology"="obstruction" and "finding site" = "bronchial structure" and "due to"="allergic reaction"

Using the short normal form

To test if "allergic asthma" subsumes another expression the short normal form is used as the predicate. Any candidate expression that satisfies all the conditions of this candidate is subsumed by "allergic asthma".

- The concept "allergic asthma", only subsumes expressions that explicitly include a focus concept that is either "asthma" or a subtype of "asthma" and the attribute "due to"="allergic reaction".
    - The candidate expression "disease" with "morphology" = "obstruction" and "finding site" = "bronchial obstruction" and "due to" = "allergic reaction" is <u>not</u> subsumed by "allergic asthma".

[3] An intermediate primitive is a concept that is primitive but which has fully defined supertypes and subtypes.

4.5.3.4 *Normal form of a fully-defined concept with fully-defined attribute value*

The concept "neoplasm of right lower lobe of lung" Figure 7) is fully-defined with a high-level proximal primitive ("disease"). The long normal form consists of the proximal primitive supertype and all the defining relationships of "neoplasm of right lower lobe of lung". However, the value of the "finding site" attribute ("structure of right lower lobe of lung") is itself fully defined. Therefore, this value is also transformed to normal form ("structure of lower lobe of lung" with "laterality" = "right"). The short normal form is the same because all of the defining relationships differ from those of the proximal primitive supertype.

The standard SNOMED CT distribution format does not support explicit nesting of definitions. The normal forms described in this require nesting and this is supported by the SNOMED CT expression model. As a result, the normal forms shown here differ from the distributed relationships table and from the canonical table.

Figure 7. Normal forms of a fully-defined finding concept with fully defined attribute value

Concept	126716006 | neoplasm of right lower lobe of lung |
Definition *(distributed)*	116680003 | is a | = 126713003 | neoplasm of lung | {116676008 | associated morphology | = 108369006 | neoplasm | ,363698007 | finding site | = 266005 | structure of right lower lobe of lung | }
Long NF *(candidate)*	64572001 | disease | : {116676008 | associated morphology | = 108369006 | neoplasm | ,363698007 | finding site | = (90572001 | structure of lower lobe of lung | : 272741003 | laterality | = 24028007 | right |)}
Short NF *(predicate)*	*Same as long form*

Using the long normal form

To test if "neoplasm of right lower lobe of lung" is subsumed by another expression the long normal form is used as the candidate. If all the conditions of a predicate expression are satisfied by this candidate then "neoplasm of right lower lobe of lung" is subsumed by this predicate.

- The concept "neoplasm of right lower lobe of lung" is subsumed by any normal form predicate expression with a focus concept that is "disease" (or a supertype of "disease" such as "clinical finding") unless the predicate expression also has conditions that do not subsume "morphology"="neoplasm" and "finding site" = "structure of lower lobe of the lung" with "laterality"="right".

Using the short normal form

To test if "neoplasm of right lower lobe of lung" subsumes another expression the short normal form is used as the predicate. Any candidate expression that satisfies all the conditions of this candidate is subsumed by "neoplasm of right lower lobe of lung".

- The concept "neoplasm of right lower lobe of lung", only subsumes expressions that have a "finding site" that is the "right lower lobe of the lung". However, because this site is normalized an expression that post-coordinates the laterality and the site will also be subsumed.

4.6 Applying normal forms to expressions

The previous section described the manner in which normal form expression transformations are applied to concept definitions. In this section this approach is extended to cover expressions which may contain refinements or qualifications of the released concepts.

4.6.1 Normal form of a simple expression

The simplest expression consists of a reference to a single concept (i.e. a single conceptId with no refinements). The normal forms for this are the same as those for the concept definition. Figure 8 illustrates this using one of the examples used in the previous section.

Figure 8. A simple expression with no refinements

Expression view	Expression
Close-to-user	71620000 | fracture of femur |
Normal-form (short or long)	64572001 | disease | : {116676008 | associated morphology | = 72704001 | fracture | ,363698007 | finding site | = 71341001 | bone structure of femur | }

4.6.2 Normal forms of expressions with refinements

Figures 9 to 13 show several possible refinements that might be applied to this expression and illustrated the resulting normal forms.

If a refinement specifies a more specific (subtype) value for one of the defining relationships of the focus concept, the refined value simply replaces the value in the definition. The examples in 9 and 10 illustrate this for refinements to either site or the morphology.

Figure 11 extends this example to include refinements to both the morphology and site. In all these examples while the refinements were not grouped in the close-to-user form, the transformation groups the site and morphology. This occurs because the refined values are replacing the defining values.

Figure 9. An expression with a refinement to finding site

Expression view	Expression
Close-to-user	71620000 | fracture of femur | : 363698007 | finding site | 29627003 | structure of neck of femur |
Normal-form (short or long)	64572001 | disease | : {116676008 | associated morphology | = 72704001 | fracture | ,363698007 | finding site | = 29627003 | structure of neck of femur | }

Figure 10. An expression with a refinement to the nature of the morphology

Expression view	Expression
Close-to-user	71620000 | fracture of femur | : 116676008 | associated morphology | = 134341006 | displaced fracture |
Normal-form (short or long)	64572001 | disease | : {116676008 | associated morphology | = 134341006 | displaced fracture | ,363698007 | finding site | = 71341001 | bone structure of femur | }

Figure 11. An expression with a refinement to the morphology and site

Expression view	Expression
Close-to-user	71620000 | fracture of femur | : 116676008 | associated morphology | = 134341006 | displaced fracture | ,363698007 | finding site | = 71341001 | bone structure of femur |
Normal-form (short or long)	64572001 | disease | : {116676008 | associated morphology | = 134341006 | displaced fracture | ,363698007 | finding site | = 71341001 | bone structure of femur | }

4.6.3 Normal forms of expressions with qualifiers

Figure 12 shows the effect of applying a qualifier attribute to a concept. As this is a qualifier it is not present in the definition and there is no indication whether this qualifier should be grouped with the site and morphology. Therefore, the qualifier remains ungrouped in the normal form.

Figure 12. An expression with a qualifier applied to specify severity

Expression view	Expression
Close-to-user	71620000 | fracture of femur | : 246112005 | severity | = 24484000 | severe |
Normal-form (short or long)	64572001 | disease | : 246112005 | severity | = 24484000 | severe | {116676008 | associated morphology | = 72704001 | fracture | ,363698007 | finding site | = 71341001 | bone structure of femur | }

4.6.4 Normal forms of expressions with nested refinements

A refinement may be applied to a value in the expression rather than directly to the focus concept. Laterality is the most obvious example of a nested refinement and is used for all the illustrations in this section. However, nesting also occurs with other expressions, notably expressions that include explicit representations of context (see section 4.6.8).

Figure 13 illustrates this by showing the effect of applying laterality to refinement to the finding site. The resulting normal form groups the finding site and its nested laterality refinement, with the morphology because this sub-expression is a valid refinement of the defined site.

Figure 13. An expression with refinement of the laterality (nested with body structure)

Expression view	Expression
Close-to-user	71620000 | fracture of femur | : 363698007 | finding site | = (71341001 | bone structure of femur | : 272741003 | laterality | = 7771000 | left |)
Normal-form (short or long)	64572001 | disease | : {116676008 | associated morphology | = 72704001 | fracture | ,363698007 | finding site | = (71341001 | bone structure of femur | : 272741003 | laterality | = 7771000 | left |)}

4.6.5 Normal forms representations of laterality

The previous section used laterality as an illustration of normalization of nested expressions. However, there are several ways in which lateralized findings or procedures may be represented by refinement expression. These approaches are discussed in the following paragraphs and are illustrated in Figures 14 to 17.

Figure 14 shows the effect of applying laterality as a nested refinement to the finding site. This approach follows the pattern shown in Figure 13, which requires restatement of the finding site even though the value of this is unchanged. The only difference between this example and Figure 13 is that the focus concept does not have a morphology attribute and as a result there is no grouping of attributes in the normal form.

Figure 14. Nested laterality refinement

Expression view	**Expression**
Close-to-user	47933007 | foot pain | : 363698007 | finding site | = (56459004 | foot structure | : 272741003 | laterality | = 7771000 | left |)
Normal-form (short or long)	22253000 | pain | : 363698007 | finding site | = (56459004 | foot structure | : 272741003 | laterality | = 7771000 | left |)

Figure 15 shows an alternative that is available for sites where there is body structure concept that is specific for lateralized body structure. (e.g. the concept "left foot" exists and is fully defined as "foot structure" with "laterality"="left"). The value "left foot" is a subtype of "foot structure" and so can be applied directly as a refinement of finding site.

The lateralized body structure concept "left foot" is fully defined and includes the defining relationship "laterality"="left". Therefore, when this concept is transformed to its normal form the expression is identical to the nested laterality example.

Figure 15. Alternative expression refinements representing lateralization

Expression view	**Expression**
Lateralized body structure value	47933007 | foot pain | : 363698007 | finding site | = 22335008 | structure of left foot |
Normal-form (short or long)	22253000 | pain | : 363698007 | finding site | = (56459004 | foot structure | : 272741003 | laterality | = 7771000 | left |)

Figure 16 illustrates an alternative that is only available in a limited number of cases where a concept exists that pre-coordinates a finding (or procedure) with a lateralized finding site. The example shown here is artificial because the concept "pain in left foot" is not present in SNOMED CT and there are no plans to add such concepts. However, some concepts of this nature do exist and their definitions when transformed should result in the same normal form.

Figure 16. Laterality precoordinated in a finding

Expression view	**Expression**
Close-to-user	<example-only> | pain in left foot | :
Normal-form (short or long)	22253000 | pain | : 363698007 | finding site | = (56459004 | foot structure | : 272741003 | laterality | = 7771000 | left |)

Figure 17 shows a close-to-user form in which laterality has been applied directly to a finding. For the purposes of computing equivalence and subsumption the concept model always treats laterality as applying to body structures rather than directly to findings or procedures. However, a simple transform rule allows a close-to-user expression consisting of a finding with a direct laterality refinement to be normalized. This normalization rule specifies that the laterality refinement is applied to all lateralizable sites in the normalized expression. The end result of this transform is exactly the same normal form as results from other approaches. The same approach can be used for procedures and an example of this is illustrated in Figure 21.

Figure 17. Laterality applied directly to a finding

Expression view	**Expression**
Nested laterality	47933007 | foot pain | : 363698007 | finding site | = (56459004 | foot structure | : 272741003 | laterality | = 7771000 | left |)
Lateralized body structure value	47933007 | foot pain | : 363698007 | finding site | = 22335008 | structure of left foot |
Laterality applied directly to finding	47933007 | foot pain | : 272741003 | laterality | = 7771000 | left |
Normal-form (short or long)	22253000 | pain | : 363698007 | finding site | = (56459004 | foot structure | : 272741003 | laterality | = 7771000 | left |)

Conclusions on approaches to laterality

In principle all four of the above approaches should be regarded as acceptable as they can all be transformed to the same normal form. However, the forms in which laterality is pre-coordinated either with the site (Figure 15) or with the finding or procedure (Figure 16) are only available for a limited number of concepts and there are no plans to systematically extend this coverage. Therefore, the main choice for consistent use is between the nested form (see Figures 13 and 14) and the direct application of laterality to a finding or procedure concept (see Figures 17 and 21).

In terms of the formal concept model the nested form may seem more appropriate. However, the direct approach has three significant advantages for recording and communication of information.

- Where multiple sites are involved (see Figure 21) or where multiple separately grouped actions apply to the same site, this approach avoids the need to specify laterality separately for each site[4]. Routinely presenting users with a choice of which sites are to be lateralize is likely to hinder acceptance.
- The nested approach "locks-in" the site value(s) and groupings present in the definition at the time the expression is authored. If a future release enhances or corrects that definition, instances of the same refinement before and after a change will not compute as equivalent. However, if laterality is applied directly the derived normal forms will be identical irrespective of when the expression was created.

[4] Only on very rare occasions will a single finding or procedure require separate lateralization of different sites in its definition. However, support for the direct approach does not preclude the nested approach if it is necessary to associate different laterality refinement with different structures.

- The resulting expressions are simpler and more compact and the transform rules mean no information is lost.

4.6.6 Normal forms of expressions including refinements of a primitive concept

When the focus concept is primitive or has an intermediate primitive supertype the normal and close to user forms are less likely to be the same as one another.

Figure 18 illustrates the effects of a refinement applied to primitive concepts. The same general rules apply but after the transformation process the same primitive focus concept remains. The short normal form expression is identical to the close-to-user form because the refinement represents the only difference between the long normal form and the definition of the focus concept.

Figure 18. An expression that refines a primitive concept

Expression view	Expression
Close-to-user	12529006 | expiratory crackles | : 363698007 | finding site | = 303549000 | entire lower lobe of lung |
Normal-form (long)	12529006 | expiratory crackles | : 363698007 | finding site | = 303549000 | entire lower lobe of lung | ,363714003 | interprets | = 78064003 | respiratory function | ,418775008 | finding method | = (315306007 | examination by method | : {260686004 | method | = 129436005 | auscultation - action | ,363704007 | procedure site | = 257728006 | anatomical concepts |})
Normal-form (short)	12529006 | expiratory crackles | : 363698007 | finding site | = 303549000 | entire lower lobe of lung |

Figure 19 illustrates the effects of a refinement applied to a concept with an intermediate primitive supertype. The short normal form contains the "due to" attribute because this differs between the focus concept (allergic asthma) and the primitive supertype (asthma) as when as the "causative agent" specified in the refinement.

Figure 19. An expression that refines a concept with an intermediate primitive supertype

Expression view	Expression
Close-to-user	389145006 | allergic asthma | : 246075003 | causative agent | = 260147004 | house dust mite |
Normal-form (long)	195967001 | asthma | : 246075003 | causative agent | = 260147004 | house dust mite | ,42752001 | due to | = 419076005 | allergic reaction | {116676008 | associated morphology | = 26036001 | obstruction | ,363698007 | finding site | = 955009 | bronchial structure | }
Normal-form (short)	195967001 | asthma | : 246075003 | causative agent | = 260147004 | house dust mite | ,42752001 | due to | = 419076005 | allergic reaction |

4.6.7 Normal forms for refinements of concepts with more complex definitions

Some concept definitions include multiple instances of the same defining attribute. Usually these are grouped separately for example to represent a procedure that examines one body structure and removes another. When refinements are applied to these concepts a question arises as to which value is to be refined. In most cases the transform rules allow this to be determined without requiring the close-to-user expression to explicitly state the instance that is being refined. The transform rule states that an ungrouped refinement applies to any instance of the appropriate attribute that subsumes it.

Figure 20 shows the effect of this transform rule when the refinement of procedure site is a subtype of one of the defining site attributes but not of the other one. The appropriate value is refined and the other value is unchanged.

Figure 20. An expression that refines one of two sites

Expression view	Expression
Close-to-user	116028008 | salpingo-oophorectomy | : 363704007 | procedure site | = 280107002 | entire left fallopian tube |
Normal-form (short or long)	71388002 | procedure | : {260686004 | method | = 129304002 | excision - action | ,363704007 | procedure site | = (181463001 | entire fallopian tube | : 272741003 | laterality | = 7771000 | left |)} {260686004 | method | = 129304002 | excision - action | ,363704007 | procedure site | = 15497006 | ovarian structure | }

Figure 21 shows a case in which a refinement of laterality is applicable to both the sites in the definition of a procedure (i.e. both "ovarian structure" and "fallopian tube structure" are lateralizable). Therefore, the resulting normal form shows this lateralization applied to both structures.

Figure 21. An expression that lateralizes multiple sites

Expression view	Expression
Close-to-user	116028008 | salpingo-oophorectomy | : 272741003 | laterality | = 7771000 | left |
Normal-form (short or long)	71388002 | procedure | : {260686004 | method | = 129304002 | excision - action | ,363704007 | procedure site | = (15497006 | ovarian structure | : 272741003 | laterality | = 7771000 | left |)} {260686004 | method | = 129304002 | excision - action | ,363704007 | procedure site | = (31435000 | fallopian tube structure | : 272741003 | laterality | = 7771000 | left |)}

In a few cases a refinement that is a valid for more than one attribute may need to be applied specifically to just one of those attributes. In these cases, the close to user form should include an attribute group with at least one other attribute from the appropriate group in the concept definition. This allows the distinction to be made by the transform rules for attribute group merging.

Otherwise the close-to-user form should not repeat groups or additional attributes that are unchanged from the definition. These attributes and groups are derivable by the normal form transformation. However, if they are included in the stored or communicated close-to-user form they are "locked-in", which may impair equivalence testing across releases.

4.6.8 Normal forms and the context model

The SNOMED CT context model is designed to allow equivalence and subsumption testing to take account of difference in the context in which a finding or procedure concept is used. The same general transform rules apply to concepts that include explicit statements of context. However, in addition to these rules the default context to a finding or procedure expression that has no explicitly stated context. This additional step allows the equivalence and subsumption tests to be applied in exactly the same way to expressions without stated context and to those with a stated context.

Figure 22 shows an expression in which the focus concept "family history of disorder" has a definition that includes stated context. The disorder "allergy to nuts" is stated as the "associated finding". Both these concepts are transformed to their respective normal forms

- The normal form of "family history of disorder" is a context wrapped in which "person in the family" is the value of "subject relationships context".
- The normal form of "allergy to nuts" ("allergy" with "causative agent"="nut") becomes the nested value of the "associated finding" attribute of the context wrapper.

Figure 22. An expression that includes specific context information

Expression view	Expression
Close-to-user	281666001 | family history of disorder | : 246090004 | associated finding | = 91934008 | allergy to nuts |
Normal-form (short or long)	243796009 | context-dependent category | : {246090004 | associated finding | = (106190000 | allergy | : 246075003 | causative agent | = 13577000 | nut |) ,408729009 | finding context | = 410515003 | known present | ,408731000 | temporal context | = 410512000 | current or specified | ,408732007 | subject relationship context | = 303071001 | person in the family | }

Figure 23 shows an expression that does not state its context. Applying the transform rules the normal form expression is generated as in previous examples. When the additional step to apply the default context is carried out a default context-wrapper ("known present" in "the subject of the record" at "current or specified time") is created and the clinical expression becomes the clinical-kernel within this wrapper.

In this case, because the concept asthma is primitive, the short normal form excludes the morphology and finding site attributes (see 4.6.6).

Figure 23. Applying default context to an expression

Expression view	**Expression**
Close-to-user	195967001 | asthma | : 246112005 | severity | = 255604002 | mild |
Normal-form *with context* (long)	243796009 | context-dependent categories | : {246090004 | associated finding | = (195967001 | asthma | : 246112005 | severity | = 255604002 | mild | {116676008 | associated morphology | = 26036001 | obstruction | ,363698007 | finding site | = 955009 | bronchial structure | }) ,408729009 | finding context | = 410515003 | known present | ,408731000 | temporal context | = 410512000 | current or specified | ,408732007 | subject relationship context | = 410604004 | subject of record | }
Normal-form *with context* (short)	243796009 | context-dependent categories | : {246090004 | associated finding | = (195967001 | asthma | : 246112005 | severity | = 255604002 | mild |) ,408729009 | finding context | = 410515003 | known present | ,408731000 | temporal context | = 410512000 | current or specified | ,408732007 | subject relationship context | = 410604004 | subject of record | }

4.6.9 Normal forms that take account of the information model

When SNOMED CT expressions are used in record systems or electronic communication there is often some surrounding contextual information that may affect the way in which the meaning of the expression should be interpreted. For example, a reference to a disease within a part of a record dedicated to "family history" should not be interpreted as a diagnosis of the patient. This is a complex area because many different information models and conventions may apply in different systems. However, some general rules have been identified and can be defined in relation to standard reference models (e.g. the HL7 Version 3 Reference Information Model).

The general rules are that contextual information apparent in the surrounding information model should be separated from the SNOMED CT expression before it is normalized. The expression is then transformed to a normal form expression. If the resulting normal form contains a context wrapped, this is separated from the clinical-kernel. A new context wrapper is derived by merging the information model context and any context stated in any original context wrapper. The SNOMED CT default context is only applied to fill in the gaps where neither the information model nor the original wrapper provides a definitive value for a context attribute. The clinical-kernel is then nested in the new context wrapper.

The examples in this section cover two of the most common areas in which context from the information model affects the meaning of a contained expression in a predictable and processable way.

Figure 24 illustrates the fact that when a SNOMED CT expression representing a procedure exists in an information construct that represents a request, the default "procedure context" value "done" is overridden by the information model. Thus the resulting normal form expressions show the "procedure context" value "requested".

Figure 24. Information model representation of context affecting default context

Expression view	Expression
Information model	<u>Request</u> • Represented by the information model for example HL7 Observation.moodCode="RQO" 113075003 | creatinine measurement, serum |
Close-to-user *with context*	400999005 | procedure requested | : 363589002 | associated procedure | = 113075003 | creatinine measurement, serum |
Normal-form *with context* (long)	243796009 | context-dependent category | : 363589002 | associated procedure | = (252144003 | biochemical test | : 116686009 | has specimen | = (123038009 | specimen | : 370133003 | specimen substance | = 67922002 | serum |) ,246093002 | component | = 15373003 | creatinine |) ,408730004 | procedure context | = 385644000 | requested | ,408731000 | temporal context | = 410512000 | current or specified | ,408732007 | subject relationship context | = 410604004 |subject of record|
Normal-form *with context* (short)	243796009 | context-dependent category | : 363589002 | associated procedure | = 113075003 | creatinine measurement, serum | 408730004 | procedure context | = 385644000 | requested | ,408731000 | temporal context | = 410512000 | current or specified | ,408732007 | subject relationship context | = 410604004 |subject of record|

Figure 25 illustrates that when the information model applies a value to a measurement procedure the resulting statement expresses a finding (i.e. the finding of a specific value as a result of that procedure).

If the requirement is to distinguish between requested and completed procedures, the default procedure context-wrapper could be applied. This would (correctly) assert that the procedure had been done ("procedure context"="done").

However, if the requirement is to distinguish between goals and actual measured values the default finding context-wrapper is more appropriate. This would indicate that this was a finding that was known to be present ("finding context"="known present").

Figure 25. Measurement procedures with values assigned in the information model

Expression view	**Expression**
Information model	Report • Represented by the information model for example HL7 Observation.moodCode="EVN" 113075003 | creatinine measurement, serum | **Value = 16 g/L** • Represented by the information model for example by HL7 Observation.value
Normal-form *with context* (short)	243796009 | context-dependent categories | : {246090004 | associated finding | = 113075003 | creatinine measurement, serum | ,408729009 | finding context | = 410515003 | known present | ,408731000 | temporal context | = 410512000 | current or specified | ,408732007 | subject relationship context | = 410604004 | subject of record | } **Value = 16 g/L** • Represented by the information model for example by HL7 Observation.value

5 Transforming expressions to normal forms

5.1 Overview

This section provides a step by step guide to the process of transforming a valid close-to-user expression into a normal form.

Figure 26 illustrates an overview of the process of normalization of an expression. Subsequent sections describe the processes shown in this diagram.

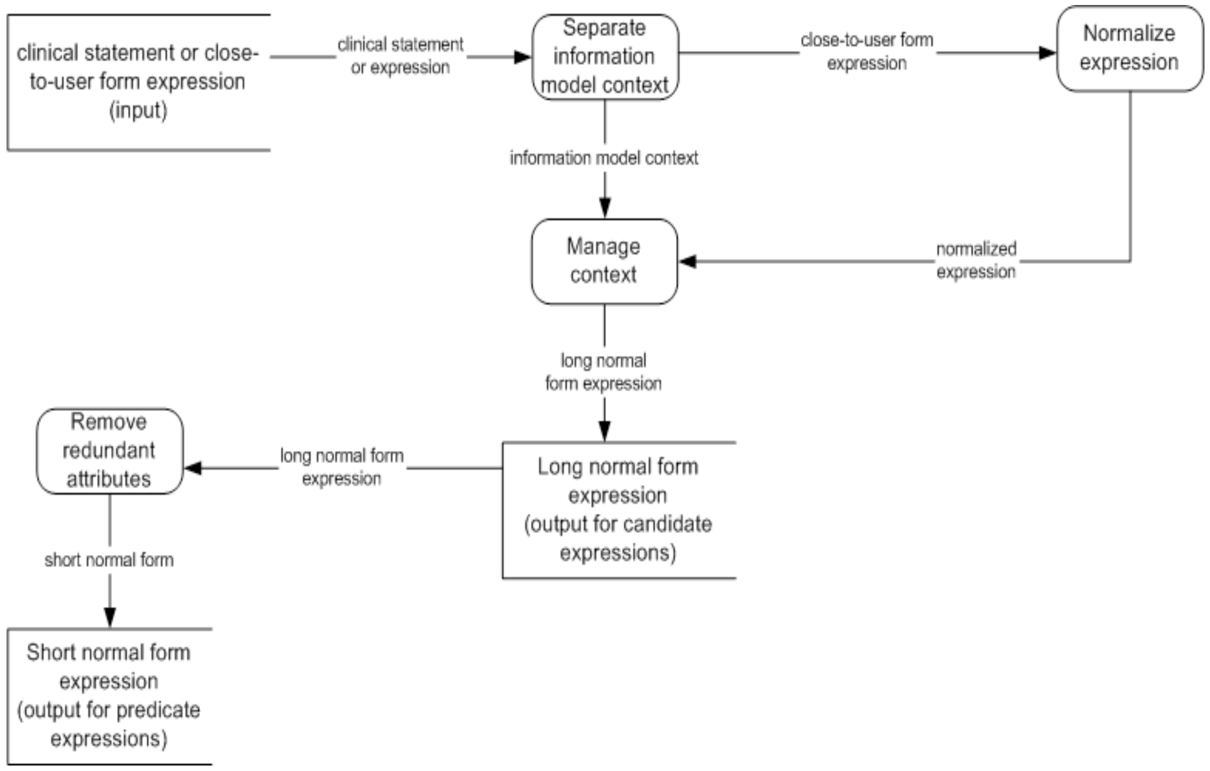

clinical statement or close-to-user form expression (input)
clinical statement or expression
Separate information model context
close-to-user form expression
Normalize expression
information model context
Manage context
normalized expression
long normal form expression
Remove redundant attributes
long normal form expression
Long normal form expression (output for candidate expressions)
short normal form
Short normal form expression (output for predicate expressions)

Figure 26. Overview of expression normalization process

5.2 Separate information model context

The objective of this process is to separate information associated with a SNOMED CT expression from the expression itself.

Information that is not part of the SNOMED CT expression itself, may influence its interpretation.

> For example, a SNOMED CT expression used in an HL7 Observation in goal mood (moodCode="GOL") implies that the finding context "goal" applies to the expression rather than the default value "known present".

If the input is a SNOMED CT expression without any information about its use within a specific information model:

- The expression is passed unchanged to the "Normalize expression" process.
- No information model context is passed to the "Manage context" process.

If the input is an HL7 clinical statement (or a similar structure that conveys additional contextual information):

- The expression is separated from the surrounding information model information and is passed to the "Normalize expression" process.
- Relevant surrounding information model information is passed to the "Manage context" process.

The items of surrounding information that are relevant vary according to the information model and the guidelines on its use. For example, if the HL7 clinical statement model is used, any of the following attributes and related classes that are present are relevant to normalization of the expression in context:

- Act.moodCode,
- Observation.value,
- Act.negationInd,
- Act.uncertaintyCode,
- participation associations or an Act (especially "subject").

The way in which these attributes may affect SNOMED context is discussed in section 5.4.

5.3 Normalize expression

Figure 27 illustrates the steps in the process of normalizing an expression. This process takes place after separating the expression from any surrounding information model context and before managing context representation in the expression.

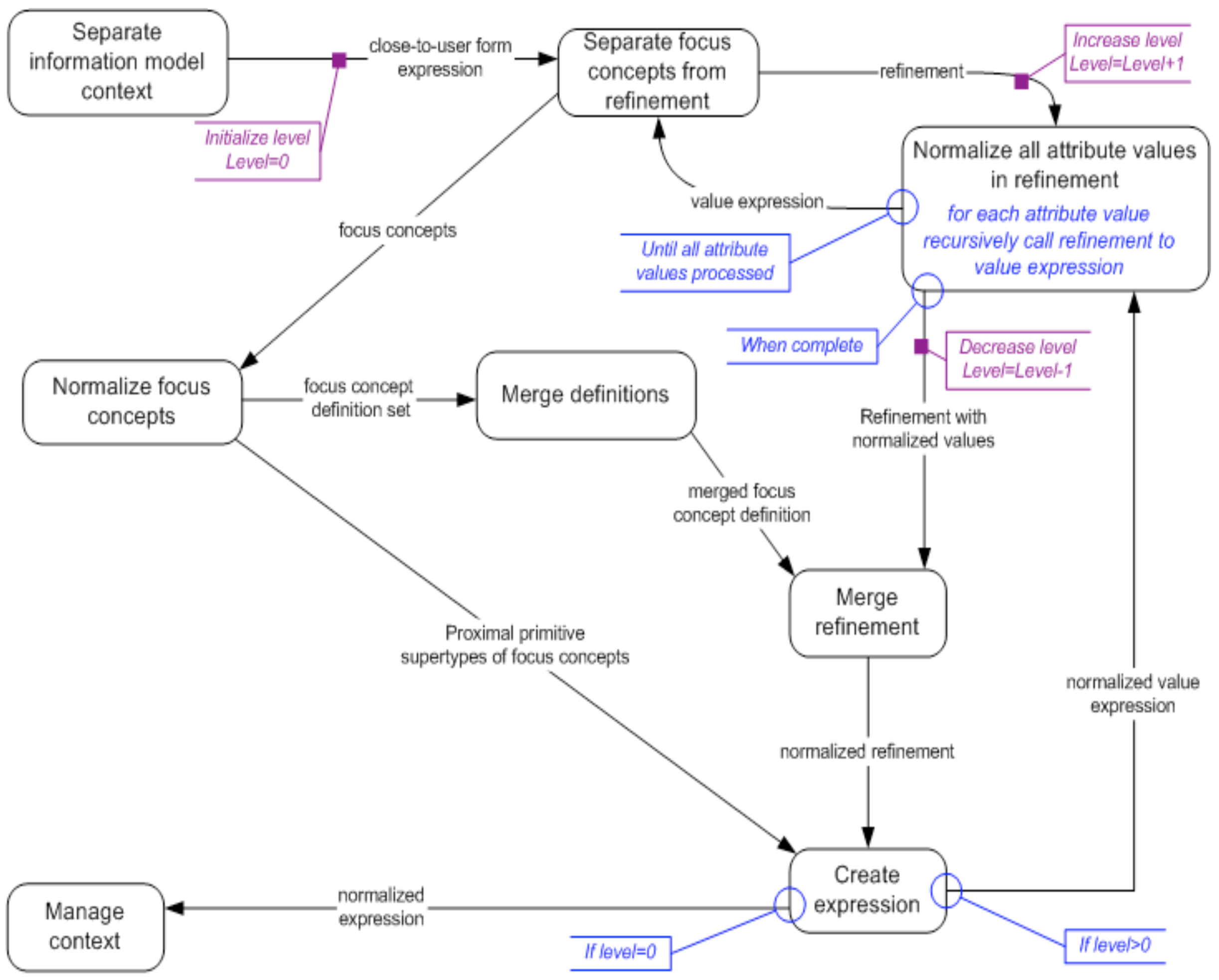

Separate information model context
close-to-user form expression
Initialize level Level=0
Separate focus concepts from refinement
refinement
Increase level Level=Level+1
Normalize all attribute values in refinement
for each attribute value recursively call refinement to value expression
value expression
Until all attribute values processed
focus concepts
When complete
Decrease level Level=Level-1
Normalize focus concepts
focus concept definition set
Merge definitions
Refinement with normalized values
merged focus concept definition
Merge refinement
Proximal primitive supertypes of focus concepts
normalized value expression
normalized refinement
Create expression
Manage context
normalized expression
If level=0
If level>0

Figure 27. Expression normalization processes

5.3.1 Separate focus concepts from refinement

The set of focus concepts in the expression is passed to the "Normalize focus concepts" process (5.3.3).

If the expression contains a refinement, this is passed to the "Normalize attribute values in refinement" process (5.3.2).

5.3.2 Normalize attribute values in refinement

The value of every attribute specified in the expression refinement (including grouped and ungrouped attributes) is treated as an expression and normalized according to the full set of rules in section 5.3. To ensure depth-first processing, this recursive process is carried out before any other processing of the expression refinement.

Recursive normalization should be applied to all values even if they are represented by single conceptIds.

When all attribute values in the expression refinement have been processed, the refinement is passed to the "Merge refinement" process (5.3.5).

5.3.3 Normalize focus concepts

The set of focus concepts is normalized to generated two separate outputs:

5.3.3.1 The set of normalized definitions of each focus concept

The set of normalized definitions includes a separate normalized definition for each focus concept,

- The normalized definition includes
  - All ungrouped relationships
  - All relationship groups complete with contained relationships
- All relationship values are normalized by recursively following the full set of rules described in section 5.3.
  - Note: Storage of pre-computed normalized form of concept definitions simplifies this process as it removes the requirement for recursive processing of definitions at run time.

The set of normalized definitions is passed to the "Merge definitions" process (5.3.4).

5.3.3.2 The non-redundant proximal primitive supertypes of the focus concepts

The non-redundant proximal primitive supertypes of the focus concepts is the set of all primitive supertypes of all the focus concepts with redundant concepts removed.

- A concept is redundant if it is:
  - A duplicate of another member of the set
  - A supertype of another concept in the set.

The set of proximal primitive supertypes generated by this process is passed to the "Create expression" process (5.3.6) as the focus concepts for the output expression.

5.3.4 Merge definitions

5.3.4.1 Overview

The set of normalized definitions derived from the "Normalize focus concepts" process (5.3.3) are merged with one another to remove redundancy. Then the normalized refinement is merged with the pre-merged definition to create a single refinement which expresses the full set of definitions and refinements without unnecessary redundancy.

The rules applied to the merger are described below for grouped and ungrouped attributes.

Group merging is completed before applying any ungrouped relationships. This ensures that, where appropriate, ungrouped attributes are applied to the correct groups in the output.

Redundant attributes are not removed until the merger process is complete. This ensures that the full set of attributes is available to allow matching throughout the process of merging.

5.3.4.2 Attribute names and attribute hierarchies

The following sections on merging groups and attributes refer to "name-matched" attributes. Two or more attributes in a definition or expression are "name-matched" if they have the same attribute name[5].

- For example, the attribute "procedure site"="appendix structure" is name-matched by the attribute "procedure site"="entire femur".

However, consideration also needs to be given to hierarchical relationships between different "attribute names". For example, "procedure site – direct" and "procedure site – indirect" are subtypes of "procedure site".

The simplest approach that can be consistently applied is to treat attributes that have subsumed names as name-matched for the purposes of group and value merging. The more specific attribute name is then applied to the merged attribute in the target definition. This means that the same rules apply for merging the values of "procedure site" and "procedure site – direct" as apply to mergers of attributes with identical names and that the name "procedure site - direct" would then be applied to any values that were merged in this way.

Progress note

Review of a number of practical examples suggests that there may be some unexpected consequences of this approach. For this reason, while the issues that arise are studied further, implementers are recommended only to merge literal name-matched attributes.

Some potential issues are noted here

As definitions are refined over time there will be more use of the specific "procedure site – indirect" and "procedure site – direct". Should pre-existing refinements to the more general "procedure site" be assigned to whichever of the more specific attributes has a value that subsumes the refined value?

If this rule is applied to some combined procedures then the merger collapses some existing definitions that contain both a "procedure site" and a "procedure site - direct" so that only one of these attributes remains. This will become less of an issue as "procedure site – indirect" is applied more widely.

---

[5] The words "attribute" and "attribute name" are used here as documented in the SNOMED CT guide to the "Abstract Logical Models and Representation Forms". In SNOMED CT distribution files a "defining relationship" is equivalent to this use of the word "attribute" and a "relationship type" represents an "attribute name".

5.3.4.3 Merging groups

- If a group in one definition meets the following criteria in relation to a group in the other definition then the groups are merged:
    - At least one attribute in one of the groups is name-matched by an attribute in the other group.

      and
    - For each name-matched pair of attributes, the value of that attribute in one group either subsumes or is identical to the value of the name-matched attribute in the other group.
- Groups that meet the criteria for merging are merged by adding all attributes present in both source groups to the same group in the merged target definition.
- Groups that cannot be merged are created as separate groups in the target definition.

Note that these conditions allow additional attributes that are not name-matched to be present in either of the candidate groups. They also allow values of name-matched attributes to be subsumed in different directions between the two groups (i.e. do not require the entire of one group to be subsumed the other group).

5.3.4.4 Merging ungrouped attributes

- If an ungrouped attribute in one definition is name-matched by a grouped attribute in the other definition, this attribute is merged according to the following rules:
    - If the value of the ungrouped attribute subsumes value of the name-matched grouped attribute
        - omit the ungrouped attribute from the target definition
    - If the value of the grouped attribute subsumes the value of the name-matched grouped attribute
        - add the ungrouped attribute to the group containing the matching grouped attribute in the target definition
        - if this condition is met by multiple groups
            - add the ungrouped attribute to all groups that meet this condition
    - If the value of the name-matched grouped and ungrouped attributes are disjoint
        - add the ungrouped attribute as an ungrouped attribute in the target expression.
- If an ungrouped attribute is name-matched with an ungrouped attribute in the other definition this attribute is merged according to the following rules:
    - If the value of one of the name-matched attributes subsumes the other value
        - include the attribute with the most specific value (not grouped)
        - omit the attributed with the less specific value
    - If the value of the name-matched attributes are identical
        - Include one and omit the other
    - If neither of the of the two preceding conditions apply
        - include both attributes (not grouped),
- If an attribute is ungrouped in one expression and there is no name-matched attribute in the other definition
    - include the attribute (not grouped).

5.3.4.5 Remove redundant elements from the merged definition

Check each group in the target definition and, within that group, compare the values of any name-matched attributes.

- If an attribute in the group has a value that subsumes the value of another name-matched attribute in the same group, remove that attribute from this group in the target definition.

Check the ungrouped set of attributes.

- If any ungrouped attribute has a value that subsumes the value of a name-matched attribute, remove this ungrouped attribute from the target definition.

Note

The removal of redundancies described only applies to name-matched pairs of attributes. It does *not* affect attributes that are redundant *only* because they are present in the definitions of the primitive focus concepts. Supertype (“is a”) relationships are ignored during this stage of processing.

5.3.4.6 Completion of the definition merging

Once the focus concept definitions have been merged, the target definition is passed to the "Merge refinement" process (5.3.5).

5.3.5 Merge refinement

The normalized expression refinement from the "Normalize attribute values in refinement" process (5.3.2) is merged with the combined definition from the "Merge definitions" process (5.3.4).

The rules for this process are the same as those for merging definitions (see 5.3.4) with the following additions.

5.3.5.1 Normalization of laterality

If an attribute representing a value for "laterality" (272741003) is present in the refinement and is applied to a focus concept that is not subsumed by "body structure" (123037004), the laterality attribute should be applied to any and every lateralizable "body structure" specified in the resulting refinement.

5.3.5.2 Normalization of non-context attributes applied in a context wrapper

If the focus concept is subsumed by "context-dependent categories" (364629017) and any attributes other than valid context attributes[6] are present in the refinement, these attributes are applied as additional refinement of the value of the "associated finding" (246090004) or "associated procedure" (363589002) attribute.

5.3.5.3 Completion of the definition merging

Once the refinement has been merged the resulting final refinement is passed to the "Create expression" process (5.3.6).

5.3.6 Create expression

Create expression process combines the proximal primitive supertypes from "Normalize focus concepts" process (5.3.3) – as the new focus concepts – with the refinement derived from the "Merge refinement" process (5.3.5).

The resulting expression is now fully normalized but context information may need to be adjusted or applied by the "Manage context" process (5.4).

[6] The only valid context attributes are: "associated finding" (246090004), "associated procedure" (363589002), "finding context" (2470590016), "procedure context" (2470591017), "temporal context" (2470592012) and "subject relationship context" (2470593019).

5.4 Manage context

Figure 28 illustrates the steps involved in managing context information extracted from the input statement or expression.

The input to this process consists of the information model context, derived from the "Separate information model context" process (5.2), and the normalized expression, generated by the "Create expression" step (5.3.6) at the end of the "Normalize expression" process (5.3).

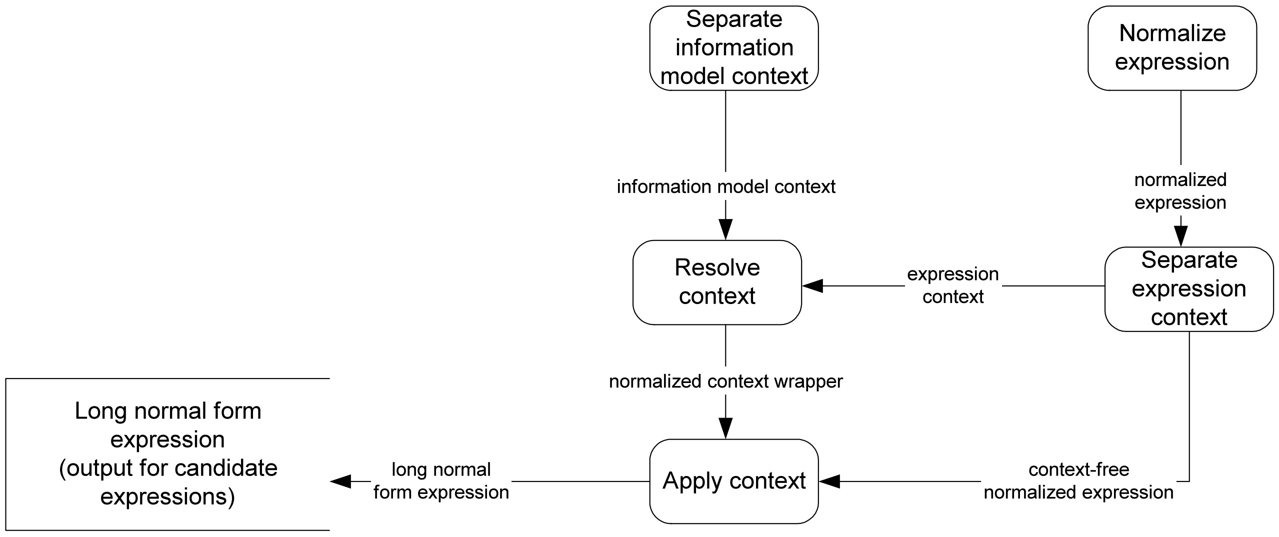

Separate information model context
Normalize expression
information model context
normalized expression
Resolve context
expression context
Separate expression context
normalized context wrapper
Long normal form expression (output for candidate expressions)
long normal form expression
Apply context
context-free normalized expression

Figure 28. Managing context in normalized expressions

5.4.1 Separate expression context

This normalized expression (generated by the "Create expression" process 5.3.6) may or may not contain any context information. If it does, this context is separated from the expression so that it can be validated and reconciled with any information model context.

If the focus concept is subtype of "context-dependent categories" (243796009)

- The expression that represents the value of the associated finding(s) (246090004) or associated procedure(s) (363589002) attribute is passed as the context-free expression to the "Apply context" process (5.4.3).
- The focus expression without the associated finding/procedure value is passed to the "Resolve context" process (5.4.2)

If the focus concept is not a subtype of "context-dependent categories" (243796009) but its refinement contains values for one or more of the following context attributes: "finding context" (2470590016), "procedure context" (2470591017), "temporal context" (2470592012) *or* "subject relationship context" (2470593019).

- These attributes are passed to the "Resolve context" process (5.4.2)
    - If the attributes present do not include a "finding context" or "procedure context" value, then an indication of the top level supertype of the focus concept is also passed with these context attributes.
- The focus expression, with the context attributes removed, is passed as the context-free expression to the "Apply context" process (5.4.3).

If neither of the above conditions apply then

- An indication of the top level supertype of the focus concept is passed to the "Resolve context" process (5.4.2)
- The entire expression is passed as the context-free expression to the "Apply context" process (5.4.3).

5.4.2 Resolve context

The resolve context process takes the information model context derived from the "Separate information model context" process (5.2) and the expression context derived from "Separate expression context" process (5.4.1) and attempts to resolve them to generate a single consistent context.

The context information in the expression or information model may unequivocally indicate that:

- Finding context applies
    - Subtypes of "finding" or "linkage concept"
    - Subtypes of "procedure" or "observable" with an associated "value" in the information model.
    - Finding context attribute value present in the expression context information.
- Procedure context applies
    - Subtypes of "procedure" or "observable" without an associated "value" in the information model.
    - Procedure context attribute value present in the expression context information.

The appropriate soft default context applies unless modified

- By specific context attributes in the expression context information
- By rules associated with particular information model context information
    - For example, *rules that are present in a reference file such as the MoodMap.xml (see* Figure 29*).*

The output is one of the following

- A single context wrapper that is passed to the "Apply context" process (5.4.3).
- An indication that context is not relevant to the expression and should not be applied. This is also passed to the "Apply context" process (5.4.3) allowing it to return a context-free expression.
- A report of errors arising from incompatibilities in the context information from the two sources.

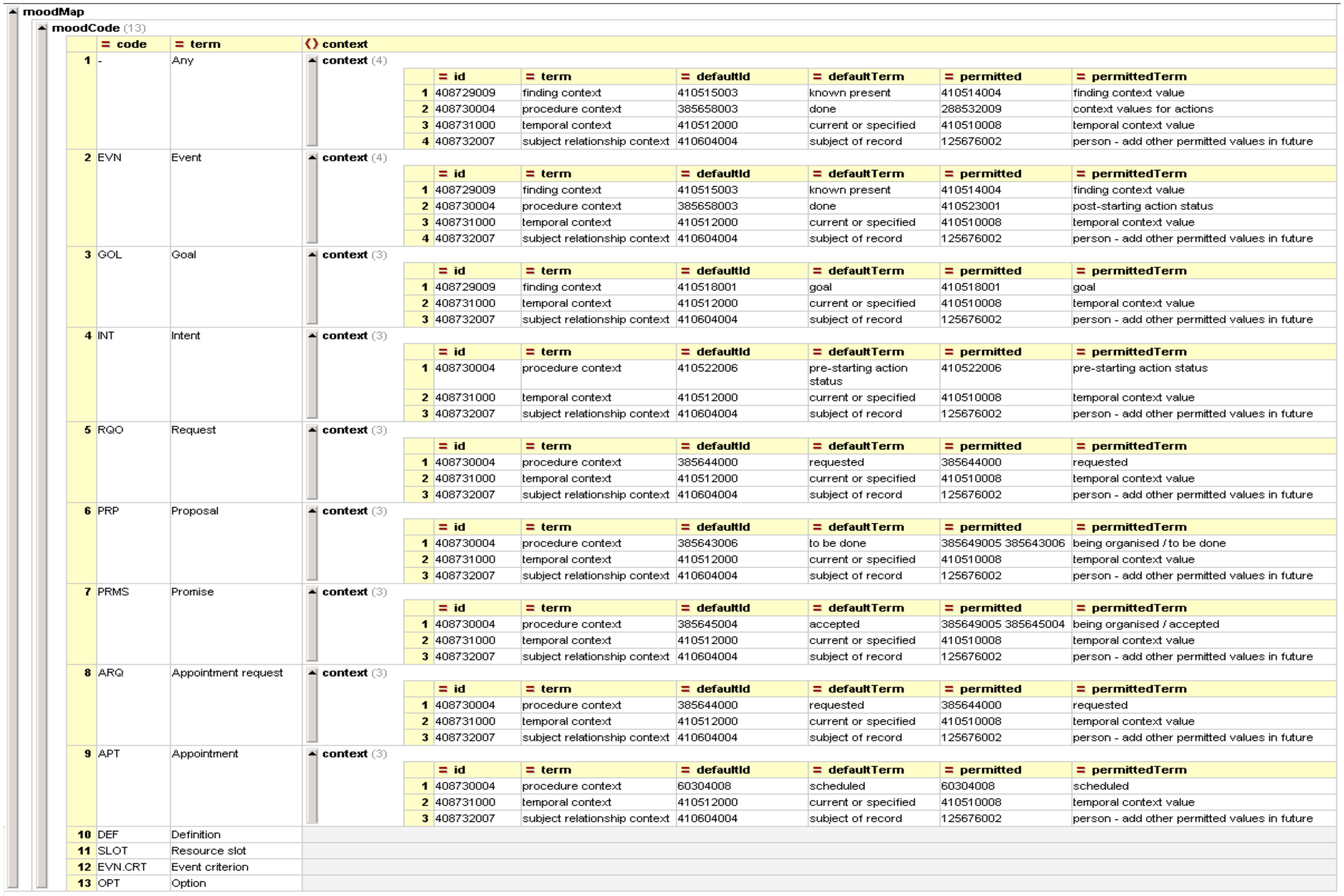
5.4.3 Apply context

If no context wrapper is provided by the "Resolve context" process (5.4.2), the context-free expression from "Separate expression context" process (5.4.1) is returned as the fully normalized expression.

If the "Resolve context" process (5.4.2) provides a context wrapper including a "finding context" (2470590016) attribute, the context-free expression from the "Separate expression context" process (5.4.1) is applied to this as the value of the "associated finding" (246090004) attribute. The resulting context-dependent expression is returned as the fully normalized expression.

If the "Resolve context" process (5.4.2) provides a context wrapper including a "procedure context" (2470591017) attribute, the context-free expression from the "Separate expression context" process (5.4.1) is applied to this as the value of the "associated procedure" (363589002) attribute. The resulting context-dependent expression is returned as the fully normalized expression.

Figure 29. MoodMap.xml file image



moodMap

moodCode (13)

	code	term	context
1	-	Any	context (4)
2	EVN	Event	context (4)
3	GOL	Goal	context (3)
4	INT	Intent	context (3)
5	RQO	Request	context (3)
6	PRP	Proposal	context (3)
7	PRMS	Promise	context (3)
8	ARQ	Appointment request	context (3)
9	APT	Appointment	context (3)
10	DEF	Definition	
11	SLOT	Resource slot	
12	EVN.CRT	Event criterion	
13	OPT	Option	

1 - Any — context (4)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408729009	finding context	410515003	known present	410514004	finding context value
2	408730004	procedure context	385658003	done	288532009	context values for actions
3	408731000	temporal context	410512000	current or specified	410510008	temporal context value
4	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

2 EVN Event — context (4)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408729009	finding context	410515003	known present	410514004	finding context value
2	408730004	procedure context	385658003	done	410523001	post-starting action status
3	408731000	temporal context	410512000	current or specified	410510008	temporal context value
4	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

3 GOL Goal — context (3)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408729009	finding context	410518001	goal	410518001	goal
2	408731000	temporal context	410512000	current or specified	410510008	temporal context value
3	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

4 INT Intent — context (3)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408730004	procedure context	410522006	pre-starting action status	410522006	pre-starting action status
2	408731000	temporal context	410512000	current or specified	410510008	temporal context value
3	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

5 RQO Request — context (3)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408730004	procedure context	385644000	requested	385644000	requested
2	408731000	temporal context	410512000	current or specified	410510008	temporal context value
3	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

6 PRP Proposal — context (3)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408730004	procedure context	385643006	to be done	385649005 385643006	being organised / to be done
2	408731000	temporal context	410512000	current or specified	410510008	temporal context value
3	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

7 PRMS Promise — context (3)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408730004	procedure context	385645004	accepted	385649005 385645004	being organised / accepted
2	408731000	temporal context	410512000	current or specified	410510008	temporal context value
3	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

8 ARQ Appointment request — context (3)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408730004	procedure context	385644000	requested	385644000	requested
2	408731000	temporal context	410512000	current or specified	410510008	temporal context value
3	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

9 APT Appointment — context (3)

	id	term	defaultId	defaultTerm	permitted	permittedTerm
1	408730004	procedure context	60304008	scheduled	60304008	scheduled
2	408731000	temporal context	410512000	current or specified	410510008	temporal context value
3	408732007	subject relationship context	410604004	subject of record	125676002	person - add other permitted values in future

Subject to change based on work in the HL7 TermInfo project

5.5 Additional steps for alternative forms

The processes described in the preceding sections generate the "long normal view". This is the most general normal form. It can be directly applied to meet key requirements related to subsumption testing. It can also be used as the source from which to derive other useful forms that are optimized for particular purposes. The following sections outline some of these.

5.5.1 Deriving the short normal view

The short normal form can be derived from the long normal form by the following steps.

5.5.1.1 Generate the set of normalized definitions of each primitive focus concepts

The set of normalized definitions includes a separate normalized definition for each primitive focus concept in the normalized expression.

- The normalized definition includes
  - All ungrouped relationships
  - All relationship groups complete with contained relationships
- All relationship values are normalized by recursively following the full set of rules described in section 5.3.
  - Note: Storage of pre-computed normalized form of concept definitions simplifies this process as it removes the requirement for recursive processing of definitions at run time.

Note this process is equivalent to the process described in 5.3.3.1 but applies to the primitive focus concepts in a normalized expression rather than to the initial set of focus concepts.

5.5.1.2 Merge the generated definition sets

This process follows exactly the same rules as the "Merge definitions" process (5.3.4)

5.5.1.3 Removed redundant attributes and groups

Attributes and groups shared with the merged definition are removed from the refinement. Only groups and ungrouped attributes that are identical can be removed from the refinement. If a group is not identical the parts that are similar cannot be removed.

5.5.1.4 Recursive removal of redundancy

The process described in this section is recursively applied to any nested expressions that remain after the top-level process to remove redundant attributes and groups.

Unlike the process of normalization, this process is done breadth first at each level in the hierarchy. If long normalized forms at nested levels are shortened before checking for redundancy, the expression will not match those in the merged definition even if they are semantically identical.

5.5.2 Canonical representations

The idea of a canonical representation is that it generates a predictable string rendering. The missing element to deliver this in the description of the "long normal form", is a specified sort order within the collections elements in an expression. A standard sort order is not essential for general purpose use but it is very useful to enable fast matching of logically identical expressions (which might otherwise be obscured by differences in order that have no semantic relevance).

The canonical form for a SNOMED CT expression is regarded as being the long normal form ordered according to the following sorting rules.

- The expression is rendered in the form specified by the SNOMED CT compositional grammar. For canonical representation a restricted version of the compositional grammar is used:
    - No whitespace characters may be included in the canonical form
    - No pipe characters "|" and thus no term text shall be included in the canonical form.
    - Thus the only permitted characters are:
        - Digits [0-9] – for conceptId values.
        - Plus [+] – to combine focus concepts.
        - Colon [:] – to represent the start of a refinement.
        - Equals [=] – to link an attribute name to it value
        - Comma [,] – to separate attributes within a refinement
        - Round brackets [()] – to represent nesting.
        - Curly brackets [{}] – to represent grouping.
- The syntax determines the general order of elements within an expression as follows.
    - Focus conceptIds;
    - Attributes (expressed as name-value pairs);
    - Groups (containing attributes).
- Within a set of focus conceptIds
    - ConceptIds are sorted alphabetically based on their normal string rendering (i.e. digits with no leading zeros)
        - The reason for alphabetic sorting rather than numeric sorting is that it is complex to sort attributes and groups which consist of an arbitrary number of conceptIds using numeric keys.
- Within a set of ungrouped attributes or a set of attributes within a group
    - Attributes are sorted alphabetically based on the string concatenation of the name and value conceptIds separated by an "=" sign.
    - If a value contains nested refinements, the value is enclosed in round brackets (which may influence the sort order) and the elements of the nested expression are sorted by applying the general canonical sorting rules.
- Within a set of groups
    - Groups are sorted by alphabetical order of the combined set of previously sorted attributes.

6 Testing subsumption and equivalence between expressions

6.1 Introduction

The main reason for generating normal form expressions is to enable testing for equivalence and subsumption between different post coordinated expressions. This section describes how these processes are carried out.

The process of generating normal form for an expression also requires testing of subsumption between subsidiary elements within the expression.

6.2 Testing for equivalence

The following steps can be applied to test for equivalence between any two valid expressions.

1. Transform both expression to long normal form (see 5.2 to 5.4)
2. Render these normal forms according the canonical representation (see 5.5.2)
3. Perform a simple string comparison between the two long normal forms in canonical representation.
    a. If the strings are identical then the expressions being tested are equivalent
    b. If the strings are not identical the two expressions being tested are not logically equivalent.

Note that this does not prove that the expressions are not equivalent. This limitation applies for the following reasons:

- One or more of the concepts referenced by the original expressions may be primitive (i.e. not fully defined) and this may obscure the equivalence[7].
- Two expressions may include an alternative "sufficient set" of attributes that imply the same meaning[8].

[7] This issue will gradually diminish in significance as more concepts are fully-defined through addition of new defining relationships.

[8] This issue will be addressed in the future as SNOMED evolves to include recognition of the distinction between "necessary" and "sufficient" sets of defining attributes. The SNOMED document on "*Abstract Logical Models and Representational forms*" includes a detailed discussion of this topic in section 2.2.5 "Nature of the definition".

6.3 Testing expression subsumption

6.3.1 Overview of expression subsumption testing

The following steps can be applied to test for subsumption of any *candidate expression* by a *predicate expression*.

1. Transform the predicate expression to short normal form[9] (see 5.3 and 5.5.1)
   - The resulting "predicate short normal form expression" is referred in subsequent steps as the *normalized-predicate*.
2. Transform the candidate expression to long normal form (see 5.2 - 5.4)
   - The resulting "candidate long normal form expression" is referred in subsequent steps as the *normalized-candidate*.
3. Test for subsumption between the *normalized-predicate* and the *normalized-candidate* by applying the tests described in section 6.3.2.
   - The *predicate expression* subsumes the *candidate expression* if the *normalized-predicate* subsumes the *normalized-candidate*.

6.3.2 Testing subsumption between two normal form expressions

The following steps are applied to test if a *normalized-predicate* subsumes a *normalized-candidate*. This assumes that these normal form expressions have been generated in accordance with 6.3.1.

1. Test that each **focus concept** referenced in the *normalized-predicate* subsumes at least one focus concept in the *normalized-candidate.*
   - If not, the *normalized-predicate* does not subsume the *normalized-candidate*. No further testing is required.
     - ➔ Exit with result *false*.
   - The approach to testing concept subsumption is described in section **Error! Reference source not found.**
2. Test that each **attribute group** in the *normalized-predicate* subsumes at least one attribute group in the *normalized-candidate*.
   - If not, the *normalized-predicate* does not subsume the *normalized-candidate*. No further testing is required.
     - ➔ Exit with result *false*.
   - The approach to testing attribute group subsumption is described in section 6.3.3
3. Test that each **ungrouped attribute** in the *normalized-predicate* subsumes at least one attribute (either grouped or ungrouped) in the *normalized-candidate*.
   - If not, the *normalized-predicate* does not subsume the *normalized-candidate*.
     - ➔ Exit with result *false*.
   - The approach to testing attribute subsumption is described in section 6.3.4
4. If all these tests succeed, the *normalized-predicate* subsumes the *normalized-candidate*.
     - ➔ Exit with result *true*.

[9] The *predicate long normal form* can be used instead of the *predicate short normal form*. However, the short form is preferred as it reduces the number of steps required in testing each candidate expression.

6.3.3 Testing subsumption between two attribute groups

The following steps test if a *predicate-attribute-group* subsumes *candidate-attribute-group*.

1. Check the *predicate-attribute-group* for the presence of the attribute: “finding context” (408729009).
    - If the group does not contain this attribute, apply the normal attribute group tests specified in section 6.3.3.1.
2. If the *predicate-attribute-group* contains the “finding context” (408729009) attribute, check whether its value is one of the following: “known absent” (410516002) or “definitely not present” (410594000).
    - If the attribute exists and has one of these values, apply the tests for a **context attribute group with absent finding**, as specified in section 6.3.3.2.
    - If the attribute exists and has any other value, apply the tests for a **normal attribute group**, as specified in section 6.3.3.1.

6.3.3.1 Testing a normal attribute group

The following step tests most attribute groups. However, a modified approach (see 6.3.3.2) is required in the case of attribute groups that indicate the absence of a finding.

1. Test that each **attribute** in the *predicate-attribute-group* subsumes at least one attribute in the *candidate-attribute-group*.
    - If not, the *predicate-attribute-group* does not subsume the *candidate-attribute-group*.
        - ➔ Exit with result *false*.
    - The approach to testing attribute subsumption is described in section 6.3.4
2. If all attributes in the group pass this test then the *predicate-attribute-group* subsumes the *candidate-attribute-group*.
        - ➔ Exit with result *true*.

6.3.3.2 Testing a context attribute group with absent finding

The following steps test most attribute groups that indicate the absence of a finding. This approach differs from the general tests applicable to other attribute groups because of the way in which assertions of absence affect the direction of subsumption. This is discussed in detail in Annex A.

1. Attempt to match each **attribute** in the *predicate-attribute-group* with an attribute which has the same name in the *candidate-attribute-group*.
    - If any attribute in the *predicate-attribute-group* is not matched by an attribute with same name in the *candidate-attribute-group,* the *predicate-attribute-group* does not subsume the *candidate-attribute-group*.
        - ➔ Exit with result *false*.
2. For each of the matched attributes identified in the previous step, compare the value of the attribute in the *predicate-attribute-group* with the value of the same attribute in the *candidate-attribute-group*.
    - If the attribute name is “finding context” (408729009) or “temporal context” (408731000), the *candidate-value* must be equivalent to or subsumed by the *predicate-value*.

- However, if the attribute name is "associated finding" (246090004) or "subject relationship context" (408732007), the direction of the test is inverted. In these cases, the *predicate-value* must be equivalent to or subsumed by the *candidate-value*.
- If any of these tests fail, the *predicate-attribute-group* does not subsume the *candidate-attribute-group*.
    - ➔ Exit with result *false*.
- Attribute values are expressions and are tested in the same way as any other expression (see 6.2 and 6.3).
    - Expression subsumption testing is recursive where expressions include nested qualifiers.

3. If all the tests above are successful, the *predicate-attribute-group* subsumes the *candidate-attribute-group*.
    - ➔ Exit with result *true*.

6.3.4 Testing attribute subsumption

The following steps test if a *predicate-attribute* subsumes a *candidate-attribute*.

1. Test that the candidate attribute name is either the same as or subsumed by the predicate attribute name.
    - If not, the *predicate-attribute* does not subsume the *candidate-attribute*
        - ➔ Exit with result *false*.
    - The approach to testing concept subsumption is described in section 6.3.5
2. Test that the *candidate-attribute* value is equivalent to or subsumed by the *predicate-attribute* value.
    - If not, the *predicate-attribute* does not subsume the *candidate-attribute*
        - ➔ Exit with result *false*.
    - Attribute values are expressions and are tested in the same way as any other expression (see 6.2 and 6.3).
        - Expression subsumption testing is recursive where expressions include nested qualifiers.
3. If both the above tests are successful, the *predicate-attribute* subsumes the *candidate-attribute*.
    - ➔ Exit with result *true*.

6.3.5 Testing concept subsumption

The following steps test if a *predicate-concept* subsumes a *candidate-concept*.

1. Test if candidate-concept is an inactive concept
    - *candidate-concept*.conceptStatus NOT IN (0, 6, 11)

        - If the *candidate-concept* is inactive then look for an active concept related by a historical relationship “SAME AS” or “REPLACED BY” and treat this as the *candidate-concept* in subsequent steps[10].

2. Test if the *candidate-concept* is identical to the *predicate-concept*.
    - If *candidate-concept.*conceptId == *predicate-concept.*conceptId the concepts are identical.
        - ➔ Exit with result *true* (accept equivalent)
3. Test if the *predicate-concept* is one of the supertype ancestors of the *candidate-concept*.
    - This is true if a sequence of “is a” relationships leads from the *candidate-concept* (as source – conceptId1) to the *predicate-concept* (as the target – conceptId2).
        - ➔ Exit returning the result of this test
    - Various approaches to optimization of this test are described in the SNOMED CT Technical Implementation Guide. The recommended approach using a “transitive closure table” is summarized in Section 7.

[10] Optionally active concepts that are related to ambiguous candidate-concepts by “MAY BE A” historical relationships could also be tested. However, this requires a decision as to whether the prime objective of retrieval is “completeness” (in which case include these possible related concepts) or “precision” in which case they should be excluded.

7 Optimizing concept subsumption testing

7.1 Introduction

Rapid and efficient computation of whether a concept is a subtype descendant of another concept is essential for effective transformation of expressions and for testing subsumption between expressions.

7.2 Approaches to concept subsumption testing

The SNOMED CT Technical Implementation Guide discusses several strategies for delivering efficient computation of subsumption between concepts. These are briefly summarized here with a brief evaluation of their suitability.

7.2.1 Recursive testing of subtype relationships

It is possible to determine whether one concept subsumes another concept by recursively following every possible sequences of “is a” Relationships from a candidate concept until the predicate concept is reached or until all possible paths have been exhausted.

This approach is far too slow to deliver effective implementations in all environments in which it has been tested to date.

7.2.2 Semantic type identifiers and hierarchy flags

Flags added to the internal representation of each Concept can be used to indicate the set of high-level concept nodes of which that concept is a subtype. A concept can only subsume concepts that include the same set of high-level concept flags. This approach can reduce the number of tests that need to be performed to recursively test the subtype relationships:

- If a candidate does not have all the high-level node flags that the predicate has, no further tests are needed. The candidate is not a subtype of the predicate.
- Even if a candidate shares the high-level node flags with the predicate, any path that reaches a concept that does not share those flags need not be further tested.

While faster than the unaided recursive testing approach, this is too slow to deliver effective implementations and is not scalable.

7.2.3 Use of proprietary database features

Some databases include additional features to support the recursive testing of a chain of hierarchical relationships. Other methods of optimization that may be applied to allow more rapid computation of subtype descendant relationships are outlined in the following subsections.

Current experiences of databases that support this type of approach indicate that (while easy to implement) the performance is substantially inferior to use of branch-numbering or transitive closure (see 7.2.5).

7.2.4 Branch numbering

The internal representation of each *Concept* can be extended to include a branch-number and a set of branch-number-ranges. A branch-numbering algorithm can be applied when each release of *SNOMED CT* is imported with each concept allocated a branch number and a set of one or more ranges including the branch numbers of all subtype concepts. At run time, rather than needing to traverse many subtype *Relationships* the branch-number of each *Concept* is tested for inclusion in the branch number range of the putative ancestor.

This approach can deliver high-performance results (e.g. 10,000 tests per second on a 2GHz PC). However, it is less flexible than transitive closure requiring a rebuild of the branch numbers and ranges each time a concept is added or when definitions are updated.

7.2.5 Precomputed Transitive Closure table

The transitive closure table approach is not included in the Technical Implementation Guide at this stage but is described in the document on “SNOMED CT Abstract Logical Model and Representational Forms”.

The “transitive closure” is a comprehensive view of all the supertypes of every concept. It can be derived from current release data by traversing all "is a" relationships recursively and adding each inferred supertype relationship to a table. Applied to the current content of SNOMED CT, it results in a table containing about six million rows.

The advantage of this type of view is that a *candidate-concept* can be tested for subsumption by *predicate-concept* by a simple SQL query. In addition, the table can be updated to take account of changes without requiring a complete rebuild. The disadvantage is the storage capacity required for this table, which (including indexes) is of the order of 1-2 Gigabytes based on the current release of SNOMED CT.

7.2.6 Recommendations

The Transitive Closure method is strongly recommended for use in any environment requiring high performance where disk capacity for storage and/or bandwidth for distribution are not a problem.

Where disk capacity and/or distribution bandwidth are limiting factors, Branch Numbering provides an efficient alternative approach.

7.3 Transitive closure implementation

7.3.1 Transitive closure distribution

It has been proposed that in future a transitive closure table would be released. This would support easier implementation and provide a reference against which to check alternative algorithms. The format for such a distributed transitive closure table is the subject of separate documents that are under development.

The following sections provide basic advice on generating a using a simple and functional transitive closure table. Even when the core SNOMED CT transitive closure is distributed there will be a continuing requirement for many implementers to support computation of the transitive closure to include additional content in Extensions to SNOMED CT.

7.3.2 Transitive closure table structure

The simplest form for a transitive closure table has two columns labeled “SubtytpeId” and “SupertypeId”. Each of these columns has a datatype that supports the SNOMED CT Identifier and is populated by concept identifiers.

This simple table requires one unique index “SubtypeId+SupertypeId” and a secondary non-unique index by “SupertypeId” to allow efficient reversed lookup.

Additional columns may be included to optimize some extended functionality. For example:

- o A flag to indicate rows that represent links between a concept and its proximal primitive supertypes.
- o If inactive concepts are included in the table, a flag to indicate the nature of any historical relationship traversed.

- A semantic distance count indicating the number of direct "is a" relationship between the subtype and supertype. Although such a number has not absolute meaning it may be useful as a relative measure of proximity.
- An identifier of the transitive closure row. This may be of value for maintaining history of changes to transitive closures between releases.

7.3.3 Generating a transitive closure table

There are various ways in which a transitive closure table can be generated. The method illustrated here represents the smallest SQL query that might be used for this purpose. It is not the most efficient query and may take several hours to run. However, it does offer a simple standard for comparing the results of alternative approaches.

```
-- Create simple sct_transitive closure table
-- This version uses varchar for string ConceptId values
-- An alternative would be to use bigint using the 64-bit integer representation
CREATE TABLE [sct_transitiveclosure] (
        [SubtypeId] [varchar] (18) COLLATE Latin1_General_CI_AS NOT NULL ,
        [SupertypeId] [varchar] (18) COLLATE Latin1_General_CI_AS NOT NULL ,
        CONSTRAINT [PK_sct_transitiveclosure] PRIMARY KEY  CLUSTERED
        (
                [SubtypeId]
                [SupertypeId],
        )
)
```

```
-- Add inverted index
CREATE  INDEX [IX_sct_transitiveclosure] ON [dbo].[sct_transitiveclosure]([SupertypeId])
```

```
-- Initial row creation self reference
INSERT INTO [sct_transitiveclosure]([SupertypeId], [SubtypeId])
SELECT [ConceptId],[ConceptId] FROM [sct_concepts]
```

```
-- Further row creation

WHILE @@ROWCOUNT>0
-- Repeats while new rows are created by the following query
INSERT INTO [sct_transitiveclosure] ([SupertypeId], [SubtypeId])
SELECT distinct [ConceptId2],[tc].[SubtypeId] FROM [sct_relationships] INNER JOIN
[sct_transitiveclosure] as tc
ON [ConceptId1]=[tc].[SupertypeId]
LEFT OUTER JOIN  [sct_transitiveclosure] As tc2
ON [ConceptId2]=[tc2].[SupertypeId] AND [tc].[SubtypeId]=[tc2].[SubtypeId]
WHERE [RelationshipType]='116680003' AND [tc2].[SubtypeId] is null
```

7.3.4 Using the transitive closure table to check subsumption

The following SQL query illustrates a simple way to use the transitive closure table for testing subsumption. In practice, this type of clause would be included in a more complex query allowing many candidates and predicates to be tested as a condition of retrieval in a single query.

```
SELECT * FROM TransitiveClosure AS tc
        WHERE [tc].[Supertype]= [precidicate-concept].[conceptId]
                AND [tc].[SubtypeId]=[candidate-concept].[conceptId]
```

Unless storage capacity is a significant constraint, a pre-computed transitive closure table appears to out-perform other options and is robust, flexible and easy to implement.

8 Optimization of normalization and expression subsumption testing

8.1 Introduction

The steps in the normal transform and subsumption testing processes are not particularly onerous. However, queries require thousands or millions of such tests to be carried out. It is therefore likely that most practical implementations will require some type of optimization to support tests for subsumption between expressions.

The method described in this section is one approach to optimization. The central idea is the use of a repository to store expressions and relationships between expressions.

The advantages of this include the following

- All transform computations can be done off-line rather than at run-time
- Less transforms are done as each distinct candidate expression need only be transformed once when created and once more each time a new release alters the definitions on which it is based.
    - Other approaches either require
        - real-time transformation each time a candidate expression is considered for retrieval

        or

        - storage of normal forms in each record entry and updating of each normal form instance whenever a new release affects the definitions on which it is based

      Neither of these approaches appears to be scalable over time, as record volumes increase. In contrast the proposed optimization is not affected by the total number or records but only by the total number of distinct expressions encountered.
- Additional optimization is possible by pre-classifying the repository so that individual queries can test an expression with a single join to a table representing the transitive closure of all used expressions.

The approach described in the following section is only one way of implementing the central idea of optimization using an expression repository. There several alternatives ways to harness the same general technique and some of these may be better suited to particular requirements or technical environments.

8.2 Expression repository design

The primary requirements for an expression repository are:

- Allocation of a fixed length, unique identifier for every expression used in an operational environment.
  - The size of an operational environment may range from an individual application at a particular site to a large multi-site organization using multiple applications that use the same expression repository.
  - The easiest way to deliver unique identifiers within this range of organizational scales is the use of a UUID (also referred to a GUID). The UUID/GUID allocation algorithm provides an industry standard approach to allocation of universally unique 128-bit identifiers and is readily available in all widely used operating systems.
- Linking every close-to-user expression with its long normal form.
  - It is necessary to update this link each time a new release of SNOMED CT changes an underlying relationship that may affect the normal form.
- The expressions themselves need to be searchable
  - The canonical version of the SNOMED CT compositional grammar is recommended for this purpose because it has a minimum of syntactic noise and a specified sort order for the elements within the expression.
  - Despite these advantages some normal form expressions can be quite long. Currently the longest normal forms seen are up to 300 characters long. For a degree of future proofing it is suggested that the longest indexable variable length character string should be used (e.g. in MS SQL Server a length of up to 900 characters).

Two possible designs that meet this goal are suggested in next two sections.

- The first option uses two tables – one to identify expressions and the other to link them to indicate the results of transformation (see 8.2.1).
- The second approach is less flexible and in its current form it lacks many of the features of the first option. It is included in this document because it follows an original suggested design and indicates an alternative for consideration and discussion. (see 8.2.2).

Whichever of these designs is used the general steps in using the tables is similar (see 8.3)

8.2.1 Dual table expression repository

The dual table approach is more flexible and potentially more compact than the single table approach (8.2.2). The compactness is achieved because each distinct normal form only occurs once in one row of the table. The flexibility results from the ability to make multiple links between expressions to specify the results of different transforms without repeating the expression.

Each discrete expression (whether close-to-user or one of the normal forms) is allocated a unique ExpressionId when it is first used or generated. From this point on, this ExpressionId is immutably linked to the expression and the expression must not be altered in anyway.

The ExpressionLink table represents the linkage between an expression and its normal forms. ExpressionLinks can be updated as necessary (i.e. when a new release is received) without any effect on the existing content of the Expression table. However, an additional row will be added to the Expression table whenever a transform results in a new expression (i.e. any expression that is not in the Expression table).

Table 1. Suggested structure for the link table in a dual table expression repository

Expression Table			
Each row in the Expression table represents and identifies an expression. All expressions used are identified in this way (i.e. close-to-user and normal forms) and the links between an expression and a transformed version of that expression are specified by the ExpressionLink table.			
Primary key	**Field Type**	**Permitted characters**	**Length**
ExpressionId	*GUID*	**binary**-or-string version	***16***/*36*
Unique *identifier for this expression.*			
Data Fields	**Field Type**	**Permitted characters**	**Length**
Expression	*String*	**0** *to* **9** *and* **{ } ( ) - , + = :**	*6 -900*
Canonical rendering of an expression in SNOMED CT compositional grammar *Indexed*			
DateAdded	*IsoDateTime*	**binary**-or-string version	***8***/*20*
Date time of addition of this Expression to the expressions table. YYYYMMDDhhmmss+ZZ.zz			

Table 2. Suggested structure for the link table in a dual table expression repository

ExpressionLink Table			
Each row in the ExpressionLink table links a source Expression with the result of transforming that expression. The DateIn and DateOut columns allow active and inactive links to be stored in the same table – easing historical review of changes. The primary key ExpressionLinkId allows multiple rows to link the same pair of expressions where a link is valid in one release, not valid in the next and restore in a subsequent release.			
Primary key	**Field Type**	**Permitted characters**	**Length**
ExpressionLinkId	*GUID*	**binary**-or-string version	***16****/36*
Unique *identifier for this expression link*			
Data Fields	**Field Type**	**Permitted characters**	**Length**
SourceExpressionId	*GUID*	**binary**-or-string version	***16****/36*
Foreign key link to the Expression table row for the source expression which is linked to a transform by this link. ***Indexed***			
ResultExpressionId	*GUID*	**binary**-or-string version	***16****/36*
Foreign key link to the Expression table row for an expression representing the result of the transform applied to a the source expression ***Indexed***			
TransformType	*Enum*	**digits [0-9]**	*2*
An enumerated value representing the nature of the transform between the source and result expressions. Values might include: 0=Single concept expression → Long normal form; 1=Other expression → Long normal form; 2=Long normal form → Short normal form. A direct transform for each expression to short normal form could be added buy is not essential this can be achieved by traversing a type 0 or 1 link followed by a type 2 link.			
DateIn	*IsoDateTime*	**binary**-or-string version	***8****/20*
Date time of addition of this CtuExpression to the expressions table. YYYYMMDDhhmmss+ZZ.zz			
DateOut	*IsoDateTime*	**binary**-or-string version	***8****/20*
Date time at which this ExpressionLink was rendered obsolete by replacement. YYYYMMDDhhmmss+ZZ.zz			

8.2.2 Single table expression repository

The single table approach provides direct mapping between a close to user expression and a normal form. This was the original suggested design for an expressions table. However, a dual table approach provides a more efficient and more flexible solution (see 8.2.1).

Table 3. Suggested structure for a single table expression repository

CtuExpression Table			
Each row in the Expression table represents a close-to-user form expression and it relationship to a normal form expression.			
Key Fields	**Field Type**	**Permitted characters**	**Length**
ExpressionId	*GUID*	**0** *to* **9** *and* ***A*** *to* ***F*** *and* ***{ } -***	***16****/38*
Unique *identifier for this close-to-user expression*.			
Data Fields	**Field Type**	**Permitted characters**	**Length**
CtuExpression	*String*	**0** *to* **9** *and* ***{ } ( ) - , + = :***	*6 - 300*
Canonical rendering of close to user expression in SNOMED CT compositional grammar ***Indexed***			
LongNormalExpression	*String*	**0** *to* **9** *and* ***{ } ( ) - , + = :***	*6 - 900*
Canonical rendering of long normal form expression in SNOMED CT compositional grammar ***Indexed***			
DateUpdated	*IsoDateTime*	**0** *to* **9**	*20*
Date time of last update to the LongNormalExpression for this CtuExpression. YYYYMMDDhhmmss+ZZ.zz			
DateAdded	*IsoDateTime*	**0** *to* **9**	*20*
Date time of addition of this CtuExpression to the expressions table. YYYYMMDDhhmmss+ZZ.zz			

8.3 Using the expressions repository

Whichever approach is taken to the design of the repository the way in which it is used is similar.

8.3.1 Run-time data entry and inbound communications

Each time an expression is recorded (either directly or in an inbound communication) the expression is looked up in the repository. The expression is rendered using the canonical version of the SNOMED CT compositional grammar and an expression matching this string is looked for in the repository.

If the expression is not found a new row is added to the expression repository.

Whether a row is found or added the unique identifier of the expression is added to the record entry or other resource in which the information encoded by the expression is to be stored. Depending on authentication and other requirements, the original form of the expression may also be stored.

This step is often referred to a "just-in-time pre-coordination" because a pre-coordinated identifier for the post-coordinated expression is generated at the time it is required. It is also possible to prime the repository with a range of expressions that are anticipated (e.g. because they are generated by a particular set of forms or protocols).

8.3.2 Run-time display or outbound communications

Although an expression repository may be shared across a large multi-site organization, there are advantages in requiring communication to adhere to standards that are not limited to bounds of that organization and which are not dependent on real-time communication with the repository. Therefore, when there is a requirement to display or communicate the information represented by an expression identifier, the expression should be looked up in the repository and added to a communication in its original form.

8.3.3 Support for normal form transformation of new expressions

The transforms described in this document are applied to all new expressions in the repository. Where a transforms results in a new expression, this expression is added to the repository. The appropriate reference between the original expression and normal form expression is created, in a manner determined by the repository design.

8.3.4 Support for normal form transformation after updates to SNOMED CT definitions

After a SNOMED CT update, the repository is refreshed to check for consequent changes in the normal forms for existing expression. Where changes are required the appropriate rows in the repository are added or updated in accordance with the repository design.

8.3.5 Support for retrieval – basic option

Retrieval requests can be dealt with by using the repository to locate the appropriate normal forms for the predicate expression and for candidate expressions. Instead of requiring processing to transform expressions in real-time a simple SQL query can immediately return the appropriate expression.

With the basic option, the process of testing subsumption is as documented in this guide (see section 6).

8.3.6 Support for retrieval – advanced option

Further optimization is possible if the expressions in the repository are classified to generate an extended transitive closure table. This process is similar in effect to testing for subsumption between every pair of expressions in the repository and recording the results of each successful test as a row in a table that identifies the relationship between the subsuming and subsumed expression. This process may appear to be unscalable because it requires many millions of tests to be carried out. However, fortunately algorithms are available that can optimize this process and classify hundreds of thousands of concepts in a little over an hour on what would today be considered a fairly modest system.

If this approach is followed, any predicate expression can be tested against any candidate expression by a simple SQL query using this extended transitive closure table. The result of this is that testing subsumption of an expression will perform practically as fast as testing the subsumption between two pre-coordinated concepts.

8.3.7 Is storing an expression the same as creating a new concept?

In one sense adding an expression to a repository and giving it an identifier is the same as creating a new concept. However, there are some subtle but highly significant differences between storing, identifying and reusing an expression in the way suggested in the guide and a SNOMED CT concept. These are summarized in Table 4.

Table 4. Differences between concepts and stored expressions

SNOMED CT released concept	Stored expression
The defining relationships of a SNOMED CT concept are intended to represent the meaning of words or phrases as they are used in clinical practice.	An expression is the collection of references to a set of SNOMED CT concepts.
The meaning of a SNOMED CT concept is represented by the fully specified name (and sometimes by an associated textual definition).	An expression is not associated with a specific text string. It may be rendered in different human readable forms but its only source of meaning is the meaning of the concepts it references.
Because a concept definition attempts to express the human understood meaning of a word or phrase the logical definition expressed by its defining relationship may not be sufficient to fully define the concept. In these cases the concept definition is marked as "primitive".	Because an expression has no specific term or source of meaning other than the focus concept and attribute it is inherently "fully-defined" in that those attributes fully define what the expression may be used to represent.
A SNOMED CT Concept can be bound to various terms that are deemed to be synonyms in a given language. The SNOMED CT design provides a framework for managing these bindings, correcting errors, supporting translations and tracking the history of changes.	It may be tempting to associate particular words or phrases with an expression. This will inevitably occur in instances in individual record entries. However, terms should not be bound to expressions in a way that suggests a formal persisting association between that term and the class represented by the expression. If such a binding is required a SNOMED CT concept should be requested (or created in an extension) to provide a proper framework for managing that binding.

Annex A Recording and retrieving absent findings

A.1. Introduction

This Annex is based on a white paper produced in May 2006 to consider the impact of "negatives" on transformation, normalization and subtype testing rules. The outcome of this was revision of the rules on subsumption testing in relation to context attribute groups that include finding context values indicating that a finding is known to be absent (see 6.3.3).

A.2. Rationale

The wider issue of different types of negation cannot be completely resolved in a short space of time – as has been demonstrated during numerous previous discussions. However, there are some aspects of current advice on computation of subsumption that appear to be misleading in relation to concepts that express the absence of a finding.

For example, current subtype testing rules on the following expression:

```
373572006 | clinical finding absent | :
          246090004 | associated finding | = (125605004 | fracture of bone | :
                    363698007 | finding site | = 71341001 | bone structure of femur | )
```

normalizes as follows

```
243796009 | context-dependent category | :
 {246090004 | associated finding | =
  (64572001 | disease | :
    {116676008 | associated morphology | = 72704001 | fracture |
     ,363698007 | finding site | = 71341001 | bone structure of femur | } )
 ,408729009 | finding context | = 410516002 | known absent |
 ,408731000 | temporal context | = 410512000 | current or specified |
 ,408732007 | subject relationship context | = 410604004 | subject of record | }
```

The result of applying “normal” subsumption testing rules is that "no fracture of femur" is subsumed by "no fracture of bone. Superficially this may seem reasonable, but it will incorrectly cause the inference that a person with a record of "no fracture of the femur" has "no fracture of a bone". This is true if "no fracture of a bone" meant one bone that it not fractured, but the generally understood meaning would be that the patient had no fractured bones.

Thus the objective was to revise the transformation and/or subtype testing rules to appropriately handle expressions that represent absent findings.

A.3. Overview

The approach specified in this document deals with the computational issue of subtype testing based in the current concept model, classifier logic and distribution format of SNOMED CT. The approach has been tested and produces reasonable results with current data. It also works appropriate with combined presence and absence finding (e.g. "head injury without skull fracture") provided these are modelled using separate context attribute groups.

The positive statement in the previous paragraph must be tempered by the knowledge that a logical technical approach is only a part of the solution. Human factors are an important issue when considering the proper processing of concepts of absence and other forms of negation. Therefore the final part of the paper consider what people may mean when explicitly stating "absence of a finding" in a clinical statement and what other people may mean when querying the record for presence or absence of a finding.

The technical approach suggested for subsumption testing expressions that involve absence of a finding are valuable only if applied appropriately. Human interpretation may be required to determine the clinical relevance of the results of absent subtype tests for a particular purpose.

A.4. Testing subsumption of absence of a finding

A.4.1 Initial assumptions

The general rules for computation of subsumption of expressions and transformation to normal forms are stated in detail in the SNOMED CT document on transformation to normal forms. They can be summarized as follows:

When two expressions are tested for subsumption, tests are performed recursively on the following elements within the normal form of those expressions:

- Groups of attribute value pairs;
- Attribute value pairs;
- Nested expressions use to represent values within an attribute value pair.

The normal form of an expression is a derived by a set of rules which retain the full semantic meaning of the original expression while transformation it to a form in which:

- Every referenced focus concept is a primitive concept
- Every attribute value is a normalized expression
- Grouping and nesting of attributes is aligned with the concept model
- Default context or context derived from the information model is made explicit using SNOMED CT context attributes

A.4.2 Identifying expressions that include absence

The normal form of any expression that represents absence of finding includes the following standard context attributes:

```
243796009 | situation with explicit context | :
  408729009 | finding context | = 410516002 | known absent |  (or a subtype)
  ,408731000 | temporal context | = <temporal context value>
  ,408732007 | subject relationship context | = <subject relationship context value>
  ,246090004 | associated finding | = <clinical finding expression>
```

When the value of "finding context" is "known absent" (or one of its subtypes) then it may be appropriate to apply subtype testing rules based on absence. However, as discussed in section A.5 of this paper, the decision on whether the rules are appropriate to a specific query depends on the intended results.

A.4.3 Testing groups rather than expressions

The relevant information in an expression can be regarded as a group of attributes as follows.

```
  {408729009 | finding context | = 410516002 | known absent |  (or a subtype)
  ,408731000 | temporal context | = <temporal context value>
  ,408732007 | subject relationship context | = <subject relationship context value>
  ,246090004 | associated finding | = <clinical finding expression> }
```

Considering absence at the group level, rather than at the expression level, allows account to be taken of expressions that refer to presence of one finding and absence of another.

The following style of expression represents the presence of "first clinical finding" and the absence of "second clinical finding".

```
243796009 | situation with explicit context | :
{408729009 | finding context | = 410515003 | known present | (or a subtype)
  ,408731000 | temporal context | = <temporal context value>
  ,408732007 | subject relationship context | = <subject relationship context value>
  ,246090004 | associated finding | = <first clinical finding expression> }
{408729009 | finding context | = 410516002 | known absent |  (or a subtype)
  ,408731000 | temporal context | = <temporal context value>
  ,408732007 | subject relationship context | = <subject relationship context value>
  ,246090004 | associated finding | = <second clinical finding expression> }
```

In this case, the first group is tested according to the general subsumption testing rules and the approach to absence may be appropriate to the second group (i.e. the group that includes "finding context"="known absent").

The overall expression, containing both these groups, is then tested in the general way according to whether the two groups separately pass the relevant test. The general subsumption testing rules allow groups not present in the predicate expression to be present in the candidate expression. Therefore both of the following predicate expressions subsume the candidate expression above irrespective of the special rules for handling absence.

Predicate 1 – "first clinical finding present"

```
243796009 | situation with explicit context | :
{408729009 | finding context | = 410515003 | known present | (or a subtype)
  ,408731000 | temporal context | = <temporal context value>
  ,408732007 | subject relationship context | = <subject relationship context value>
  ,246090004 | associated finding | = <first clinical finding expression> }
```

Predicate 2 – "second clinical finding absent"

```
243796009 | situation with explicit context | :
{408729009 | finding context | = 410516002 | known absent |  (or a subtype)
  ,408731000 | temporal context | = <temporal context value>
  ,408732007 | subject relationship context | = <subject relationship context value>
  ,246090004 | associated finding | = <second clinical finding expression> }
```

A.4.4 Testing "associated finding" in groups containing "known absent"

If a group contains "finding context"="known absent" then the test applied to the value of the "associated finding" attribute is changed.

The general purpose test for the value of an attribute is:

- "is the candidate value identical to or a subtype of the predicate value"

The alternative test when the group contains "known absent" is:

- "is the **predicate** value identical to or a subtype of the **candidate** value"

A.4.5 Testing "subject relationship context" in groups containing "known absent"

If a group contains "finding context"="known absent" then the test applied to the value of the "subject relationship context" attribute should also be changed to the alternative form.

- "is the **predicate** value identical to or a subtype of the **candidate** value"

Thus

- "family history of heart disease in father" implies "family history of heart disease"; but
- "no family history of heart disease" implies "no family history of heart disease in father".

A.4.6 Testing "temporal context" in groups containing "known absent"

If a group contains "finding context"="known absent" then the test applied to the value of the "temporal context" attribute needs to be carefully considered depending on the intended result of the query.

In some cases it may also be changed to the alternative form.

- "is the **predicate** value identical to or a subtype of the **candidate** value"

Thus

- "currently has asthma" implies "current has, or at some time in past had, asthma"; but
- "did not have headache recently" does not imply "did not have headache in the past".

However, since the value "all times past" is specified for expressing concepts like "never had a headache" the standard subsumption test rules may work better in some cases.

Since the time aspect in the record is relative to the time of recording while the intended result of a query may be relative to a specified time (or the time of the query) the use of temporal context in queries requires careful consideration on a query by query basis.

A.4.7 Differences between subject relationship and temporal context

The difference between the handling of "subject relationship context" and "temporal context" noted in A.4.5 and A.4.6 may result from a significant difference in value hierarchies.

Thus "no family history of asthma" literally means something like:

"As far as is known, at <u>all times in the past</u>, the disorder asthma was absent from, <u>all members of the subjects family</u> "

The temporal context value hierarchy includes the value "all times past" to capture one part of this. However, for the "all members of the subject's family" we use the same "member of family" concept as is used for asserting "at least one member of the family".

An argument can be made for aligning the approach in both these hierarchies in one of two ways:

a) Removing the value "all times past" from "temporal context" and using "current or past" in its place. Then the subtype testing of temporal context would invert in the same way as for the other attributes (i.e. in absence mode the "current or past" would imply all other temporal context … aka "all times past").
b) Adding "all members of family" to the subject relationship value hierarchy and carefully applying this in all negation expressions. In this case, the alternative subtype testing would only apply to "associated finding".

While approach (b) may appear more rational it does seem to have two disadvantages:

- It requires more disciplined use in modelling and in post-coordination

- Several new "all" values would be needed – "all members of paternal family", "all male members of family", "all known contacts", etc. to allow negatives to be expressed clearly.

A.4.8 Impact of nesting context expressions

Currently, the concept model does not allow for nesting of context rich expressions. However, some potential use cases have been advanced for allowing a "finding with explicit context" to be nested as the value of another attribute. If this is permitted then inclusion of "known absent" in such nested expressions will clearly have some impact.

It may be that simply applying the specific general or alternative rule at appropriate nested levels will have the desired result. However, until there are real cases to test the possibility of new exceptions arising cannot be ruled out.

A.5. Human factors and testing absence

A.5.1 Subsumption testing for classification

When consider subsumption testing as part of the process of classifying the concepts in SNOMED CT the underlying assumptions is that the comparison process is potentially symmetrical. Thus any two concepts can be compared to ask the following questions:

- Are A and B identical? … if not then
- Is A a subtype of B? … if not then
- Is B a subtype of A?

If not then we might possibly be interested in the semantic proximity of the concepts for example …

- What supertypes do A and B share?
- Are there any concepts that are subsumed by both A and B

In this relatively abstract environment it is possible to discuss ideas about "known absent" or "not done". These ideas may seem theoretically sound while being less readily applicable in practical clinical applications. In some cases the practical view may be more complex than the abstract view but in the case of "absence" it seems possible that considering real use cases may in some ways simplify or at least assist in prioritization.

A.5.2 Subsumption testing for querying records

Subsumption testing in a clinical application is typically concerned with testing instances of expressions in clinical records ("candidate expressions") against sets of criteria some of which are represented as SNOMED CT expressions ("predicate expressions").

- A predicate is an expression against which other expressions are tested. Predicate expression may be constructed for specific queries or may be developed as reusable part of clinical protocols, decision support rules or report specifications. In these cases, the author of a predicate is someone trying to find out something by querying a record or set of records.
- The candidate is an expression that is tested to see if it is subsumed by the predicate. Candidate expressions may be constructed directly by the author of a clinical statement (i.e. an instance of an entry in the record) or by an application designer determining the way in which particular user decisions are recorded. Thus the direct or indirect author of the candidate is typically someone wishing to record (or enable the recording of) a finding or procedure in a record. Although the candidate expression is a crucial part of subsumption testing its reason for existing is not determined by the requirements of a specific query but rather by what the user wishes to record.

The more abstract subsumption testing for classification described in the previous section is a prerequisite for effective subsumption testing in clinical applications. However, the differences between the motivations of those constructing predicate and candidate expressions mean that subsumption testing in clinical applications is rarely a symmetrical comparison. The typical test is "does this candidate satisfy the criteria?" or in some cases "could this candidate possibly satisfy the criteria?"

When considering absence or other kinds of negation the difference between the perception of the author of an instance of clinical information and the view of the person constructing a query may be even more significant. Thus technical rules for testing subsumption of "known absent" finding are only one part of the picture.

To avoid misunderstanding and consequent errors it is worth considering two general questions:

- What are the possible motivations for recording a "known absent" finding?
- What are the possible motivations for specifying retrieval queries for absent findings?

The next two sections identify several different answers to these questions.

A.5.3 Motivations for recording a "known absent" finding?

There thousands of possible findings that might be made at every encounter (and theoretically every second). The vast majority of absent finding are not recorded but there are clearly some good reasons for explicitly recording the absence of some findings. These might include

a) To record that the author asked a question and got a negative response

   Example: "Family history – No family history of asthma"

   "I asked the patient if anyone in their family has or had asthma and they said 'no'"

b) To record that the author examined/investigated and did not find this

   Example: "No heart murmur"

   "I listened for a heart murmur and did not hear one"

c) To record a possible conclusion that the author considered and rejected.

   Example: "Not meningitis" (as part of an assessment of a patient with a fever and headache)

   "I considered the possibility of meningitis and rejected it".

d) To refute a statement made by someone else.

   Example: "Not appendicitis" (in a record that contains an earlier assertion of "diagnosis appendicitis")

   "The admitting doctor's diagnosis of appendicitis seems to be incorrect".

e) To indicate a change in an earlier assessment

   Example: "Carcinoma of bronchus excluded" (as part of record in which the same author previously thought this a likely diagnosis)

   "I thought they might have Ca bronchus but following investigations I have now rejected this diagnosis".

f) To note the resolution of finding that was previously present.

   Example: "No abdominal pain" (in a record which has a previous finding of "abdominal pain present")

   "The abdominal pain present on admission has now resolved".

g) To indicated that a finding that is commonly present in associated with another finding is not present in this case.

   Example: "No loss of consciousness" (in the record of patient who has had a head injury)

In (a), (b) and (c) the dominant motive may be to assert what was done or considered. However, recording absent findings may also be a part of the process the author followed to organize her thoughts.

Both (d) and (e) record difference in view related to some previous assertion. Where this is the intention a strong case can be made for linking the statements in the record structure. However, this is a possible motivation even if such links are supported by the system or have not been added to this instance.

In the case of (f) the use of absent indicates a change in condition of the patient rather than an update of the diagnosis or interpretation by the clinician.

In case (g) an absent finding is recorded to refine the nature of a specific condition.

There is considerable overlap between these different motivations for recording absence. However, the overall point motivation for recording an absent finding may or may not be aligned with the reason for retrieving negative findings.

A.5.4 Motivations for specifying retrieval of "known absent" finding?

When querying for absence of a finding the most likely motivation is to establish the absence of a finding. In the absence of evidence to the contrary the normal assumption is that an abnormal finding is absent. Furthermore, in most cases a point in time assertion of absence does not imply the finding was never true, nor that could not be true at a future point in time. Thus in most cases, a query for absence is more concerned with checking that there is no statement indicating the presence of the finding rather that searching for a statement of presence.

There are exceptions to this:

- If the abnormal finding is usually found in associated with a confirmed finding
  - E.g. "absence of chest pain" in a patient with a confirmed "myocardial infarction").
- If the abnormal finding is obscure and may easily have been overlooked.
  - E.g. "Kopliks spots"
- If an assertion of presence of a finding was made by an informer of unknown reliability.
  - E.g. Bystander asserts that patient had a "heart attack" but clinical assessment excludes this.

If these exceptions apply the presence of a statement of an absent finding may be of interest. However, this depends of specific thinking around the question being posed so that the query criteria achieve the desired result. It is not enough to simply search for a specific absence and its subtypes. The assumptions about presence or absence of a finding must be considered.

> Example: Determine the number of road accident victims who have been admitted to hospital but have no fractured bones. In practical terms the best approach would just be to exclude those with known presence of fracture. The assumption is "unless a fracture is mentioned they are not known to have one".

Another possible motivation for looking for absence findings is to monitor or audit the delivery of care and check that appropriate questions have been asked, tests done, possibilities considered, etc. In these cases, the query needs to search for both presence and absence … or possibly for a procedure code representing the appropriate examination or investigation.

Example: Were all patients admitted for routine surgery asked if they had any allergies.

A.6. Conclusions on absent findings

There are rational reasons for wishing to record and retrieve information about absent findings. However, there is not a direct one-to-one relationship between the motivations for recording absence and the motivations for retrieval.

The suggested technical advice on subsumption testing of known absent findings addresses the logical question of subsumption, but this is only one part of the picture. The meaning implied by recording a known absent finding needs to be considered in the context of the intention of a query. When this is understood the alternative subsumption test can be applied appropriately to support complete and accurate retrieval.

A.7. Procedures "not done"

The use of the "procedure context" value "not done" (and subtypes) has similarities to the "finding context" value "known absent". The same alternative rules for subsumption computation could be applied to "associated procedure" value. Similar human factor considerations are also likely to apply.

The range of procedure context values is wider and covers decision, request and intent as well as the simple observation that something was not done. Thus variants such as "not to be done" and "not requested" also need to be considered.