Appendix A: mapRule Grammar and Formatting

```
Rule = TruthStatement / Clause
Clause = (ClauseFinding / ClauseObservable) [ ws ANDOP ws (ClauseFinding / ClauseObservable) ] ws
TruthStatement = ws 1*1("true" / "otherwise true") ws
;;; A Rule is either a truth statement or a clause
;;; A truth statement is either "true" or "otherwise true"
;;; A clause is either a clause with a finding or a clause with an observable
;;; and value optionally followed by the AND operator and a clause with a finding or observable and value.
ClauseObservable = "IFA" ws ( AttributeObservable ws NumericOperator ws Value )
;;; Observable clause has a mandatory value
ClauseFinding = "IFA" ws ( AttributeFinding )
;;;
AttributeObservable = ConceptObservable
;;; This could be removed and AttributeObservable changed to
;;; ConceptObservable without affecting the grammar
AttributeFinding = ConceptFinding
;;; This could be removed and AttributeFinding changed to
;;; ConceptFinding without affecting the grammar
NumericOperator = ("<" / ">=")
;;;Age at onset rules use greaterthanorequals for lower bounds and lessthan for upper bound
Value = ConceptAny / Numeric / OtherText
;;; Added in ConceptAny, which allows any Concept to be used as a Value,
;;; although clearly not all concepts are suitable
ConceptObservable = SctId ws pipe ws FullySpecifiedNameObservable ws pipe
ConceptFinding = SctId ws pipe ws FullySpecifiedNameFinding ws pipe
ConceptAny = SctId ws pipe ws FullySpecifiedName ws pipe
SctId = 6*18( digit )
;;; The SctId must be a valid SNOMED CT Concept.id value
FullySpecifiedNameObservable = 1*nonwsnonpipe *( ( 1*SP "(" *SP 1*nonwsnonparennonpipe *SP ")" !( ws pipe )
) / ( 1*SP 1*nonwsnonparennonpipe !( ws pipe ) ) ) *SP 1*1( "(observable entity)" )
;;; The FSN of an observable must have a semantic tag = "observable entity"
;;; and may contain other embedded parenthesised strings. The ! (NOT)
;;; look-ahead operator serves to prevent the parser consuming the
;;; semantic tag when it's looking for words before the tag.
```

```
FullySpecifiedNameFinding = 1*nonwsnonpipe *( ( 1*SP "(" *SP 1*nonwsnonparennonpipe *SP ")" !( ws pipe ) ) /
( 1*SP 1*nonwsnonparennonpipe !( ws pipe ) ) ) *SP 1*1( "(finding)" / "(disorder)" )
;;; The FSN of a finding must have a semantic tag = "disorder" or
;;; "finding" and may contain other embedded parenthesised strings.
;;; The ! (NOT) look-ahead operator serves to prevent the parser consuming
;;; the semantic tag when it's looking for words before the tag.
FullySpecifiedName = 1*nonwsnonpipe *( ( 1*SP "(" *SP 1*nonwsnonparennonpipe *SP ")" !( ws pipe ) ) / ( 1*SP
1*nonwsnonparennonpipe !( ws pipe ) ) ) *SP 1*1( "(" SemanticTag ")" )
;;; Any FSN must have a semantic tag and may contain other embedded
;;; parenthesised strings. The ! (NOT) look-ahead operator serves to
;;; prevent the parser consuming the semantic tag when it's looking for
;;; words before the tag.
Numeric = 1*(digit) !( *ws /OtherText )
OtherText = 1*(nonwsnonsemicolonnonpipe) *( 1*SP 1*nonwsnonsemicolonnonpipe )
;;; OtherText is used in Value and may not contain a semicolon because
;;; semicolon is the AND operator and follows a Value. Note that FSNs in
;;; Values may containg semicolons.
SemanticTag = 1*(nonwsnonparennonpipe) *( 1*SP 1*nonwsnonparennonpipe )
;;; A Semantic Tag may consist of words separated by whitespace, but may contain whitespace.
digit = %x30-39
ws = *( SP / HTAB / CR / LF )
SP = %x20
HTAB = %x09
CR = %x0D
LF = %x0A
pipe = %x7C
nonwsnonparennonpipe = %x21-27 / %x2A-7B / %x7D-7E / UTF8-2 / UTF8-3 / UTF8-4 ; no parentheses
nonwsnonsemicolonnonpipe = %x21-3A / %x3C-7B / %x7D-7E / UTF8-2 / UTF8-3 / UTF8-4 ; no parentheses
nonwsnonpipe = %x28-29 / nonwsnonparennonpipe
ANDOP = "AND" ;
UTF8-2 = %xC2-DF UTF8-tail
UTF8-3 = %xE0 %xA0-BF UTF8-tail / %xE1-EC 2( UTF8-tail ) / %xED %x80-9F UTF8-tail / %xEE-EF 2( UTF8-tail )
UTF8-4 = %xF0 %x90-BF 2( UTF8-tail ) / %xF1-F3 3( UTF8-tail ) / %xF4 %x80-8F 2( UTF8-tail )
UTF8-tail = %x80-BF
```