A Translation from SHACL into SCL Grammar

We present our translation $\tau(M)$ from a SHACL document M (a set of SHACL shape definitions) into our SCL grammar. The translation into SCL grammar of a document M containing a set M^S of shapes can be defined as: $\bigwedge_{s \in M^S} \tau(s)$, where $\tau(s)$ is the translation of a single SHACL shape s. Notice that M contains an element for each shape name occurring in M. If M contains a shape name s that does not have a corresponding shape definition, M^S will include the empty shape definition $s : \langle \{\}, \{\} \rangle$. Given a shape $s : \langle t, d \rangle$, its translation $\tau(s : \langle t, d \rangle)$ is defined in Table 1, where its constraint definition $\tau_d(x)$ equals $\tau(x, s)$. Note that we do not discuss implicit class-based targets, as they just represent a syntactic variant of class targets. In the reminder of this section we define how to compute $\tau(x, s)$.

As convention, we use c as an arbitrary constant and C as an arbitrary list of constants. We use s, s' and s'' as shape names, and $\bar{S}$ as a list of shape names. Variables are defined as x, y and z. Arbitrary paths are identified with r.

The translation of the constraints of a shape $\tau(x, s)$ is defined in two cases as follows. The first case deals with the property shapes, which must have exactly one value for the sh:path property. The second case deals with node shapes, which cannot have any value for the sh:path property.

$$\tau(x, s) = \top \wedge \begin{cases} \bigwedge_{\forall \langle s,y,z \rangle \in M} \tau_2(x, r, \langle s, y, z \rangle) & \text{if } \exists\, r. \langle s, \texttt{sh:path}, r \rangle) \in M \\ \bigwedge_{\forall \langle s,y,z \rangle \in M} \tau_1(x, \langle s, y, z \rangle) & \text{otherwise} \end{cases}$$

This translation is based on the following translations of node shapes triples, property shape triples and property paths.

A.1 Translation of Node Shape Triples

The translation of $\tau_1(x, \langle s, y, z \rangle)$ is split in the following cases, depending on the predicate of the triple. In case none of those cases are matched $\tau_1(x, \langle s, y, z \rangle) \doteq \top$. The latter ensures that any triple not directly described in the cases below does not alter the truth value of the conjunction in the definition of $\tau(x, s)$.

- $\tau_1(x, \langle s, \texttt{sh:hasValue}, \texttt{c} \rangle) \doteq x = \texttt{c}$.
- $\tau_1(x, \langle s, \texttt{sh:in}, C \rangle) \doteq \bigvee_{\texttt{c} \in C} x = \texttt{c}$.
- $\tau_1(x, \langle s, \texttt{sh:class}, \texttt{c} \rangle) \doteq \exists\, y. \texttt{isA}(x, y) \wedge y = \texttt{c}$.
- $\tau_1(x, \langle s, \texttt{sh:datatype}, \texttt{c} \rangle)) \doteq \mathrm{F}^{\mathrm{dt}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:nodeKind}, \texttt{c} \rangle) \doteq \mathrm{F}^{\mathrm{IRI}}(x)$ if $c =$sh:IRI; $\mathrm{F}^{\mathrm{literal}}(x)$ if $c =$sh:Literal; $\mathrm{F}^{\mathrm{blank}}(x)$ if $c =$sh:BlankNode. The translations for a c that equals sh:BlankNodeOrIRI, sh:BlankNodeOrLiteral or sh:IRIOrLiteral are trivially constructed by a conjunction of two of these three filters.
- $\tau_1(x, \langle s, \texttt{sh:minExclusive}, \texttt{c} \rangle) \doteq x > \texttt{c}$ if order is an interpreted relation, else $\mathrm{F}^{>\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:minInclusive}, \texttt{c} \rangle) \doteq x \geq \texttt{c}$ if order is an interpreted relation, else $\mathrm{F}^{\geq\texttt{c}}(x)$.

- $\tau_1(x, \langle s, \texttt{sh:maxExclusive}, \texttt{c}\rangle) \doteq x < \texttt{c}$ if order is an interpreted relation, else $\mathrm{F}^{<\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:maxInclusive}, \texttt{c}\rangle) \doteq x \leq \texttt{c}$ if order is an interpreted relation, else $\mathrm{F}^{\leq \texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:maxLength}, \texttt{c}\rangle) \doteq \mathrm{F}^{\text{maxLength}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:minLength}, \texttt{c}\rangle) \doteq \mathrm{F}^{\text{minLength}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:pattern}, \texttt{c}\rangle) \doteq \mathrm{F}^{\text{pattern}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:languageIn}, C\rangle) \doteq \bigvee_{\texttt{c}\in C} F^{\text{languageTag = c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:not}, s'\rangle) \doteq \neg\texttt{hasShape}(x, s')$.
- $\tau_1(x, \langle s, \texttt{sh:and}, \bar{S}\rangle) \doteq \bigwedge_{s'\in\bar{S}} \texttt{hasShape}(x, s')$.
- $\tau_1(x, \langle s, \texttt{sh:or}, \bar{S}\rangle) \doteq \bigvee_{s'\in\bar{S}} \texttt{hasShape}(x, s')$.
- $\tau_1(x, \langle s, \texttt{sh:xone}, \bar{S}\rangle) \doteq \bigvee_{s'\in\bar{S}}(\texttt{hasShape}(x, s') \wedge \bigwedge_{s''\in\bar{S}\setminus\{s'\}} \neg\texttt{hasShape}(x, s''))$.
- $\tau_1(x, \langle s, \texttt{sh:node}, s'\rangle) \doteq \texttt{hasShape}(x, s')$.
- $\tau_1(x, \langle s, \texttt{sh:property}, s'\rangle) \doteq \texttt{hasShape}(x, s')$.

A.2 Translation of Property Shapes

The translation of $\tau_2(x, r, \langle s, y, z\rangle)$ is split in the following cases, depending on the predicate of the triple. In case none of those cases are matched $\tau_2(x, r, \langle s, y, z\rangle) \doteq \top$.

- $\tau_2(x, r, \langle s, \texttt{sh:hasValue}, \texttt{c}\rangle) \doteq \exists y.r(x, y) \wedge \tau_1(y, \langle s, \texttt{sh:hasValue}, \texttt{c}\rangle)$
- $\tau_2(x, r, \langle s, p, \texttt{c}\rangle) \doteq \forall y.\tau_3(x, r, y)) \rightarrow \tau_1(y, \langle s, p, \texttt{c}\rangle)$, if p equal to one of the following: sh:class, sh:datatype, sh:nodeKind, sh:minExclusive, sh:minInclusive, sh:maxExclusive, sh:maxInclusive, sh:maxLength, sh:minLength, sh:pattern, sh:not, sh:and, sh:or, sh:xone, sh:node, sh:property, sh:in .
- $\tau_2(x, r, \langle s, \texttt{sh:languageIn}, C\rangle) \doteq \forall y.\tau_3(x, r, y)) \rightarrow \tau_1(y, \langle s, \texttt{sh:languageIn}, C\rangle)$.
- $\tau_2(x, r, \langle s, \texttt{sh:uniqueLang}, \text{true}\rangle) \doteq \bigwedge_{\texttt{c}\in L} \neg\exists^{\geq 2} y.r(x, y) \wedge F^{\text{lang}=c}(y)$ where $L = \{\texttt{c} \mid \texttt{c} \in C \wedge \exists s'.\langle s', \texttt{sh:languageIn}, C\rangle \in M\}$. This translation is possible because sh:languageIn is the only constraint that can force language tags constraints on literals.
- $\tau_2(x, r, \langle s, \texttt{sh:minCount}, \texttt{c}\rangle) \doteq \exists^{\geq \texttt{c}} y.\tau_3(x, r, y))$.
- $\tau_2(x, r, \langle s, \texttt{sh:maxCount}, \texttt{c}\rangle) \doteq \neg\exists^{\leq \texttt{c}} y.\tau_3(x, r, y))$.
- $\tau_2(x, r, \langle s, \texttt{sh:equals}, \texttt{c}\rangle) \doteq \forall y.\tau_3(x, r, y) \leftrightarrow \tau_3(x, \texttt{c}, y)$.
- $\tau_2(x, r, \langle s, \texttt{sh:disjoint}, \texttt{c}\rangle) \doteq \neg\exists y.\tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, y)$.
- $\tau_2(x, r, \langle s, \texttt{sh:lessThan}, \texttt{c}\rangle) \doteq \forall y, z\,.\,\tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, z) \rightarrow y < z$.
- $\tau_2(x, r, \langle s, \texttt{sh:lessThanOrEquals}, \texttt{c}\rangle) \doteq \forall y, z\,.\,\tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, z) \rightarrow y \leq z$.
- $\tau_2(x, r, \langle s, \texttt{sh:qualifiedValueShape}, s'\rangle) \doteq \alpha \wedge \beta$, where α and β are defined as follows. Let S' be the set of *sibling shapes* of s if M contains $\langle s, \texttt{sh:qualifiedValueShapesDisjoint}, \text{true}\rangle$, or the empty set otherwise. Let $\nu(x) = \texttt{hasShape}(x, s') \bigwedge_{\forall s''\in S'} \neg\texttt{hasShape}(x, s'')$. If M contains the triple

$\langle s, \texttt{sh:qualifiedMinCount}, \texttt{c}\rangle$, then α is equal to $\exists^{\geq \texttt{c}}\, y.\tau_3(x,r,y) \wedge \nu(x)$, otherwise α is equal to $\top$. If M contains the triple
$\langle s, \texttt{sh:qualifiedMaxCount}, \texttt{c}\rangle$, then β is equal to $\neg\, \exists^{\leq \texttt{c}}\, y.\tau_3(x,r,y) \wedge \nu(x)$, otherwise β is equal to $\top$.

- $\tau_2(x, r, \langle s, \texttt{sh:close}, \text{true}\rangle) \doteq \bigwedge_{\forall R \in \Theta} \neg\, \exists\, y.R(x,y)$ if Θ is not empty, where Θ is defined as follows. Let Θ^{all} be the set of all relation names in M, namely $\Theta^{\text{all}} = \{R \mid \langle x, R, y\rangle \in M\}$. If this FOL translation is used to compare multiple SHACL documents, such in the case of deciding containment, then Θ^{all} must be extended to contain all the relation names in all these SHACL documents. Let Θ^{declared} be the set of all the binary property names $\Theta^{\text{declared}} = \{R \mid \{\langle s, \texttt{sh:property}, x\rangle \wedge \langle x, \texttt{sh:path}, R\rangle\} \subseteq M\}$. Let Θ^{ignored} be the set of all the binary property names declared as "ignored" properties, namely $\Theta^{\text{ignored}} = \{R \mid R \in \bar{R} \wedge \langle s, \texttt{sh:ignoredProperties}, \bar{R}\rangle \in M\}$, where $\bar{R}$ is a list of IRIs. The set Θ can now be defined as $\Theta = \Theta^{\text{all}} \setminus (\Theta^{\text{declared}} \cup \Theta^{\text{ignored}})$.

A.3 Translation of Property Paths

The translation $\tau_3(x,r,y))$ of any SHACL path r is given by the following cases. For simplicity, we will assume that all property paths have been translated into an equivalent form having only simple IRIs within the scope of the inverse operator. Using SPARQL syntax for brevity, where the inverse operator is identified by the hat symbol ^, the sequence path $\hat{}(r_1/r_2)$ can be simplified into $\hat{}r_2/\hat{r}_1$; an alternate path $\hat{}(r_1 \mid r_2)$ can be simplified into $\hat{}r_2 \mid \hat{}r_1$. We can simplify in a similar way zero-or-more, one-or-more and zero-or-one paths $\hat{}(r^{*/+/?})$ into $(\hat{}r)^{*/+/?}$.

- If r is an IRI R, then $\tau_3(x,r,y)) \doteq R(x,y)$
- If r is an inverse path, with $r =$ "`[ sh:inversePath` R `]`", then $\tau_3(x,r,y)) \doteq R^{-}(x,y)$
- If r is a conjunction of paths, with $r =$ "`(` r_1`,` r_2`, ...,` r_n `)`", then $\tau_3(x,r,y)) \doteq \exists z_1, z_2, ..., z_{n-1}.\tau_3(x,r_1,z_1)) \wedge \tau_3(z_1,r_2,z_2)) \wedge ... \wedge \tau_3(z_{n-1},r_2,y))$
- If r is a disjunction of paths, with $r =$ "`[ sh:alternativePath (` r_1`,` r_2`, ...,` r_n `) ]`", then $\tau_3(x,r,y)) \doteq \tau_3(x,r_1,y)) \vee \tau_3(x,r_2,y)) \vee ... \vee \tau_3(x,r_n,y))$
- If r is a zero-or-more path, with $r =$ "`[ sh:zeroOrMorePath` r_1`]`", then $\tau_3(x,r,y)) \doteq (\tau_3(x,r_1,y)))^*$
- If r is a one-or-more path, with $r =$ "`[ sh:oneOrMorePath` r_1`]`", then $\tau_3(x,r,y)) \doteq \exists z.\tau_3(x,r_1,z)) \wedge (\tau_3(z,r_1,y)))^*$
- If r is a zero-or-one path, with $r =$ "`[ sh:zeroOrOnePath` r_1`]`", then $\tau_3(x,r,y)) \doteq x = y \vee \tau_3(x,r_1,y))$

B Translation from SCL Grammar into SHACL

We present here the translation μ, inverse of τ, to translate a sentence in the SCL grammar into a SHACL document. We begin by defining the translation of the property path subgrammar $r(x,y)$ into SHACL property paths:

- $\mu(R) \doteq R$

- $\mu(R^-) \doteq$ `[ sh:inversePath` R `]`
- $\mu(r^\star(x,y)) \doteq$
  `[ sh:zeroOrMorePath` $\mu(r(x,y))$ `]`
- $\mu(x = y \vee r(x,y)) \doteq$
  `[ sh:zeroOrOnePath` $\mu(r(x,y))$ `]`
- $\mu(r_1(x,y) \vee r_2(x,y)) \doteq$
  `[ sh:alternativePath (` $\mu(r_1(x,y))$`,` $\mu(r_2(x,y))$ `] )`
- $\mu(r_1(x,y) \wedge r_2(x,y)) \doteq$
  `(` $\mu(r_1(x,y))$`,` $\mu(r_2(x,y))$ `)`

The translation of the constraint subgrammar $\psi(x)$ is the following. we will use $\mu(\psi(x))$ to denote the SHACL translation of shape $\psi(x)$, and $\iota(\mu(\psi(x)))$ to denote its shape IRI. To improve legibility, we omit set brackets around sets of RDF triples, and we represent them in Turtle syntax. For example, a set of RDF triples such as "s `a sh:NodeShape ; sh:hasValue c .` " is to be interpreted as the set $\{\langle s, \texttt{rdf:type}, \texttt{sh:NodeShape}\rangle, \langle s, \texttt{sh:hasValue}, c\rangle\}$.

- $\mu(\top) \doteq$
  s `a sh:NodeShape .`
- $\mu(x = \texttt{c}) \doteq$
  s `a sh:NodeShape ;`
  `sh:hasValue c .`
- $\mu(F(x)) \doteq$
  s `a sh:NodeShape ;`
  f `c .`
  Predicate f is the filter function identified by F, namely one of the following: `sh:datatype`, `sh:nodeKind`, `sh:minExclusive`, `sh:minInclusive`, `sh:maxExclusive`, `sh:maxInclusive`, `sh:maxLength`, `sh:minLength`, `sh:pattern`, `sh:languageIn`.
- $\mu(\texttt{hasShape}(x, s')) \doteq$
  s `a sh:NodeShape ;`
  `sh:node` s' `.`
  if s' is a node shape, else:
  s `a sh:NodeShape ;`
  `sh:property` s' `.`
- $\mu(\neg\psi(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:not` $\iota(\mu(\psi(x)))$ `.`
- $\mu(\psi_1(x) \wedge \psi_2(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:and (`$\iota(\mu(\psi_1(x)))$`,` $\iota(\mu(\psi_2(x)))$`) .`
- $\mu(\exists^{\geq n} y.r(x,y) \wedge \psi(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:property [`
  `sh:path` $\mu(r(x,y))$ `;`
  `sh:qualifiedValueShape` $\iota(\mu(\psi(x)))$ `;`

```
      sh:qualifiedMinCount n ;
    ] .
```

- $\mu(\forall y.r(x,y) \leftrightarrow R(x,y)) \doteq$

```
  s a sh:NodeShape ;
    sh:property [ ;
      sh:path μ(r(x,y)) ;
      sh:equals R ;
    ] .
```

- $\mu(\neg \exists y.r(x,y) \wedge R(x,y)) \doteq$

```
  s a sh:NodeShape ;
    sh:property [ ;
      sh:path μ(r(x,y)) ;
      sh:disjoint R ;
    ] .
```

- $\mu(\forall y,z.r(x,y) \wedge R(x,z) \rightarrow y < z) \doteq$

```
  s a sh:NodeShape ;
    sh:property [ ;
      sh:path μ(r(x,y)) ;
      sh:lessThan R ;
    ] .
```

- $\mu(\forall y,z.r(x,y) \wedge R(x,z) \rightarrow y \leq z) \doteq$

```
  s a sh:NodeShape ;
    sh:property [ ;
      sh:path μ(r(x,y)) ;
      sh:lessThanOrEquals R ;
    ] .
```

We can now define the translation $\mu(\varphi)$ of a complete sentence of the φ-grammar into a SHACL document M (effectively a set of RDF triples) as follows.

- $\mu(\varphi_1 \wedge \varphi_2) \doteq \mu(\varphi_1) \cup \mu(\varphi_2))$
- $\mu(\psi_1(\mathtt{c})) \doteq \mu(\psi_1(x)) \cup$

```
  s a sh:NodeShape ;
    sh:targetNode c;
    sh:node ι(μ(ψ1(x))) .
```

- $\mu(\forall x.\ \mathtt{isA}(x,\mathtt{c}) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$

```
  s a sh:NodeShape ;
    sh:targetClass c;
    sh:node ι(μ(ψ1(x))) .
```

- $\mu(\forall x,y.\ R(x,y) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$

```
  s a sh:NodeShape ;
    sh:targetSubjectsOf R ;
    sh:node ι(μ(ψ1(x))) .
```

- $\mu(\forall x,y.\ R^{-}(x,y) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$

```
  s a sh:NodeShape ;
    sh:targetObjectsOf R ;
    sh:node ι(μ(ψ1(x))) .
```

– $\mu(\forall x.\, \texttt{hasShape}(x, \texttt{s}) \leftrightarrow \psi(x)) \;\doteq\; \mu(\psi_1(x)) \cup$
  s `a sh:NodeShape ;`
    `sh:node` $\iota(\mu(\psi_1(x)))$ `.`

C Additional Proof

Proof of Theorem 9. Similarly to the use of the `C` construct of `SCL`, a simple combination of few instances of the `O` feature allows to write the following sentence φ encoding the existence of an injective function that is not surjective. In more detail, a weaker version of the role of the counting quantifier is played here by the `O'` construct that enforces the functionality of the two relations F and G. In addition, by applying the `O` construct twice between the inverse of F and G, we are able to ensures that F^- is functional as well. Hence, the thesis easily follows.

$$
\begin{aligned}
\varphi &\doteq \texttt{isA}(0, \texttt{c}) \wedge \neg\, \exists\, x\,.\, F^-(0, x) \wedge \forall\, x\,.\, \texttt{isA}(x, \texttt{c}) \rightarrow \psi(x);\\
\psi(x) &\doteq \exists\, y\,.\, (F(x, y) \wedge \texttt{isA}(y, \texttt{c}))\\
&\quad \wedge\; \forall\, y, z\,.\, F(x, y) \wedge F(x, z) \rightarrow y \leq z \wedge \forall\, y, z\,.\, G(x, y) \wedge G(x, z) \rightarrow y \leq z\\
&\quad \wedge\; \forall\, y, z\,.\, F^-(x, y) \wedge G(x, z) \rightarrow y \leq z \wedge \forall\, y, z\,.\, F^-(x, y) \wedge G(x, z) \rightarrow z \leq y.
\end{aligned}
$$

To prove that `E O'` fragment does not enjoy the finite-model property too, it is enough to replace the last to applications of the `O` feature with the `E` formula $\forall\, y\,.\, F^-(x, y) \leftrightarrow G(x, y)$, which ensures the functionality of F^-, being G functional. □