A Translation from SHACL into SCL Grammar

We present our translation 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 with targets can be defined as: $\bigwedge_{\forall s \in M^S} \tau(s)$, where $\tau(s)$ is the translation of a single SHACL shape. The translation $\tau(s : \langle t, d \rangle)$ is defined with respect to t in Table 1, where its constraint definition d equals $\tau(x, s)$. 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. Despite shape names also being constants, we adopt the convention of using s, s' and s'' as shape names, and $\hat{S}$ 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.

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

Since we are considering non-recursive SHACL, we can assume that the process of progressively unfolding shape definitions eventually terminates, as shown in [**?**]. This translation is based on the following translations of property shapes triples, node shape triples and property paths.

A.1 Translation of Property 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 $\tau(x, s)$.

- $\tau_1(x, \langle s, \texttt{sh:hasValue}, \texttt{c} \rangle) \doteq x = \texttt{c}$.
- $\tau_1(x, \langle s, \texttt{sh:in}, C \rangle) \doteq \bigvee_{\texttt{c} \in C} x = \texttt{c}$.
- $\tau_1(x, \langle s, \texttt{sh:class}, \texttt{c} \rangle) \doteq \exists y.\text{isA}(x, y) \wedge y = \texttt{c}$.
- $\tau_1(x, \langle s, \texttt{sh:datatype}, \texttt{c} \rangle)) \doteq \text{F}^{\text{datatype}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:nodeKind}, \texttt{c} \rangle) \doteq \text{F}^{\text{nodeKind}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:minExclusive}, \texttt{c} \rangle) \doteq \text{F}^{>\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:minInclusive}, \texttt{c} \rangle) \doteq \text{F}^{\geq\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:maxExclusive}, \texttt{c} \rangle) \doteq \text{F}^{<\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:maxInclusive}, \texttt{c} \rangle) \doteq \text{F}^{\leq\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:maxLength}, \texttt{c} \rangle) \doteq \text{F}^{\text{maxLength}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:minLength}, \texttt{c} \rangle) \doteq \text{F}^{\text{minLength}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:pattern}, \texttt{c} \rangle) \doteq \text{F}^{\text{pattern}=\texttt{c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:languageIn}, C \rangle) \doteq \bigvee_{\texttt{c} \in C} F^{\text{languageTag = c}}(x)$.
- $\tau_1(x, \langle s, \texttt{sh:not}, s' \rangle) \doteq \neg\tau(x, s')$.

- $\tau_1(x, \langle s, \texttt{sh:and}, \hat{S}\rangle) \doteq \bigwedge_{s' \in \hat{S}} \tau(x, s')$.
- $\tau_1(x, \langle s, \texttt{sh:or}, \hat{S}\rangle) \doteq \bigvee_{s' \in \hat{S}} \tau(x, s')$.
- $\tau_1(x, \langle s, \texttt{sh:xone}, \hat{S}\rangle) \doteq \bigvee_{s' \in \hat{S}} (\tau(x, s') \wedge \bigwedge_{s'' \in \hat{S} \setminus \{s'\}} \neg\tau(x, s''))$.
- $\tau_1(x, \langle s, \texttt{sh:node}, s'\rangle) \doteq \tau(x, s')$.
- $\tau_1(x, \langle s, \texttt{sh:property}, s'\rangle) \doteq \tau(x, s')$.

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 $\tau_2(x, r, \langle s, y, z\rangle) \doteq \top$.

- $\tau_2(x, r, \langle s, \texttt{sh:hasValue}, \texttt{c}\rangle) \doteq \exists\, y. r(x, y) \wedge \tau_1(y, \langle s, \texttt{sh:hasValue}, \texttt{c}\rangle)$
- $\tau_2(x, r, \langle s, p, \texttt{c}\rangle) \doteq \forall\, y. \tau_3(x, r, y)) \to \tau_1(y, \langle s, p, \texttt{c}\rangle)$, 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` .
- $\tau_2(x, r, \langle s, \texttt{sh:languageIn}, C\rangle) \doteq \forall\, y. \tau_3(x, r, y)) \to$ $\tau_1(y, \langle s, \texttt{sh:languageIn}, C\rangle)$.
- $\tau_2(x, r, \langle s, \texttt{sh:uniqueLang}, \text{true}\rangle) \doteq \bigwedge_{\texttt{c} \in L} \neg \exists^{\geq 2}\, y. r(x, y) \wedge F^{\text{lang}=c}(y)$ where $L = \{\texttt{c} \mid \texttt{c} \in C \wedge \exists\, s^{'}. \langle s^{'}, \texttt{sh:languageIn}, C\rangle \in M\}$ (we can do this since `sh:languageIn` is the only constraint that can force language tags on literals). For the purposes of validation, it is enough for the set L to contain all the language tags in the graph to validate.
- $\tau_2(x, r, \langle s, \texttt{sh:minCount}, \texttt{c}\rangle) \doteq \exists^{\geq \texttt{c}}\, y. \tau_3(x, r, y))$.
- $\tau_2(x, r, \langle s, \texttt{sh:maxCount}, \texttt{c}\rangle) \doteq \neg \exists^{\geq \texttt{c}+1}\, y. \tau_3(x, r, y))$.
- $\tau_2(x, r, \langle s, \texttt{sh:equals}, \texttt{c}\rangle) \doteq \forall\, y. \tau_3(x, r, y) \Leftrightarrow \tau_3(x, \texttt{c}, y)$.
- $\tau_2(x, r, \langle s, \texttt{sh:disjoint}, \texttt{c}\rangle) \doteq \neg \exists\, y. \tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, y)$.
- $\tau_2(x, r, \langle s, \texttt{sh:lessThan}, \texttt{c}\rangle) \doteq \forall\, y, z\, .\, \tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, z) \to y < z$.
- $\tau_2(x, r, \langle s, \texttt{sh:lessThanOrEquals}, \texttt{c}\rangle) \doteq \forall\, y, z\, .\, \tau_3(x, r, y) \wedge \tau_3(x, \texttt{c}, z) \to$ $y \leq z$.
- $\tau_2(x, r, \langle s, \texttt{sh:qualifiedValueShape}, s'\rangle) \doteq \alpha \wedge \beta$, 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 $\gamma(x) = \tau(x, s') \bigwedge_{\forall\, s'' \in S'} \neg\tau(x, s^{''})$. If M contains the triple $\langle s, \texttt{sh:qualifiedMinCount}, \texttt{c}\rangle$, then α is equal to $\exists^{\geq \texttt{c}}\, y. \tau_3(x, r, y) \wedge \gamma(x)$, otherwise α is equal to $\top$. If M contains the triple $\langle s, \texttt{sh:qualifiedMaxCount}, \texttt{c}\rangle$, then β is equal to $\neg \exists^{\geq \texttt{c}+1}\, y. \tau_3(x, r, y) \wedge \gamma(x)$, otherwise β is equal to $\top$.
- $\tau_2(x, r, \langle s, \texttt{sh:close}, \text{true}\rangle) \doteq \bigwedge_{\forall\, R \in \Omega} \neg \exists\, y. R(x, y)$ if Ω is not empty, where Ω is defined as follows. Let Ω^{all} be the set of all relation names in M, namely $\Omega^{\text{all}} = \{R \mid \langle x, R, y\rangle \in M\}$. If this FOL translation is used to compare multiple SHACL documents, such in the case of deciding containment, then Ω^{all} must be extended to contain all the relation names in all these SHACL documents. Let Ω^{declared} be the set of all the binary property names $\Omega^{\text{declared}} = \{R \mid \{\langle s,$

$\texttt{sh:property}, x\rangle \wedge \langle x, \texttt{sh:path}, R\rangle\rangle\} \subseteq M\}$. Let Ω^{ignored} be the set of all the binary property names declared as "ignored" properties, namely $\Omega^{\text{ignored}} = \{R \mid R \in \bar{R} \wedge \langle s, \texttt{sh:ignoredProperties}, \bar{R}\rangle \in M\}$, where $\bar{R}$ is a list of IRIs. The set Ω can now be defined as $\Omega = \Omega^{\text{all}} \setminus (\Omega^{\text{declared}} \cup \Omega^{\text{ignored}})$. Note that, for the purposes of validation, it would be sufficient to use the single triple relation T, instead of the binary relations, and translate $\tau_2(x, r, \langle s, \texttt{sh:close}, \text{true}\rangle)$ into $\neg \exists y, z. \langle x, y, z\rangle \wedge \bigwedge_{\forall R \in \Omega^{\text{declared}} \cup \Omega^{\text{ignored}}} \neg y = R$.

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 "ˆ". Using SPARQL syntax for brevity, 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 P, then $\tau_3(x, r, y)) \doteq P(x, y)$
- If r is an inverse path, with $r =$ "`[ sh:inversePath` P `]`", then $\tau_3(x, r, y)) \doteq P^{-}(x, y)$
- 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 an approach to translate a stence 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:

- $\mu(P) \doteq P$
- $\mu(P^{-}) \doteq$ `[ sh:inversePath` P `]`
- $\mu(r^{\star}(x, y)) \doteq$
  `[ 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 (` $\mu(r_1(x, y))$`,` $\mu(r_2(x, y))$ `] )`

– $\mu(r_1(x,y) \land r_2(x,y)) \doteq$
($\mu(r_1(x,y))$, $\mu(r_2(x,y))$)

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 $\{\langle s,$ `rdf:type`, `sh:NodeShape`$\rangle, \langle s,$ `sh:hasValue`$, c\rangle\}$.

– $\mu(\top) \doteq$
  s `a sh:NodeShape .`
– $\mu(x = \mathtt{c}) \doteq$
  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`.
– $\mu(\neg\psi(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:not` $\iota(\mu(\psi(x)))$ `.`
– $\mu(\psi_1(x) \land \psi_2(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:and (`$\iota(\mu(\psi_1(x)))$`,` $\iota(\mu(\psi_2(x)))$`) .`
– $\mu(\exists^{\geq n} y.r(x,y) \land \psi(x)) \doteq$
  s `a sh:NodeShape ;`
  `sh:property [`
  `sh:path` $\mu(r(x,y))$ `;`
  `sh:qualifiedValueShape` $\iota(\mu(\psi(x)))$ `;`
  `sh:qualifiedMinCount` n `;`
  `] .`
– $\mu(\forall y.r(x,y) \leftrightarrow P(x,y)) \doteq$
  s `a sh:NodeShape ;`
  `sh:property [ ;`
  `sh:path` $\mu(r(x,y))$ `;`
  `sh:equals` P `;`
  `] .`
– $\mu(\neg \exists y.r(x,y) \land P(x,y)) \doteq$
  s `a sh:NodeShape ;`
  `sh:property [ ;`
  `sh:path` $\mu(r(x,y))$ `;`
  `sh:disjoint` P `;`
  `] .`

- $\mu(\forall y,z.r(x,y) \wedge P(x,z) \rightarrow y < z) \doteq$
  s `a sh:NodeShape ;`
  `sh:property [ ;`
  `sh:path` $\mu(r(x,y))$ `;`
  `sh:lessThan` P `;`
  `] .`
- $\mu(\forall y,z.r(x,y) \wedge P(x,z) \rightarrow y \leq z) \doteq$
  s `a sh:NodeShape ;`
  `sh:property [ ;`
  `sh:path` $\mu(r(x,y))$ `;`
  `sh:lessThanOrEquals` P `;`
  `] .`

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` $\iota(\mu(\psi_1(x)))$ `.`
- $\mu(\forall x.\ \mathrm{isA}(x,\mathtt{c}) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$
  s `a sh:NodeShape ;`
  `sh:targetClass c;`
  `sh:node` $\iota(\mu(\psi_1(x)))$ `.`
- $\mu(\forall x,y.\ P(x,y) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$
  s `a sh:NodeShape ;`
  `sh:targetSubjectsOf` P `;`
  `sh:node` $\iota(\mu(\psi_1(x)))$ `.`
- $\mu(\forall x,y.\ P^{-}(x,y) \rightarrow \psi_1(x)) \doteq \mu(\psi_1(x)) \cup$
  s `a sh:NodeShape ;`
  `sh:targetObjectsOf` P `;`
  `sh:node` $\iota(\mu(\psi_1(x)))$ `.`