April 1,2006

Translating SWRL FOL, SWRL, and OWL DL to the Universal Policy Logic
Version 0.1

ICS-16763-TR-07-004
SRI Project No. 16763
Contract No. FA8750-05-C-0230

Prepared by
Daniel Elenius
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025-3493

Prepared for
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

Acknowledgment and Disclaimer:
This material is based upon work supported by the Defense Advanced Projects Agency and the United State Air Force under Contract Number FA8750-05-C-0230.
Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the Defense Advanced Research Projects Agency and the United States Air Force.

1 Introduction

This document describes how to translate from OWL DL[1], SWRL and SWRL FOL to the Universal Policy Logic (UPL).

2 Translation of OWL DL

A UPL *policy* (Γ, Δ) consists of a well-typed context Γ and a set Δ of well-typed sentences over Γ. Intuitively, Γ contains the *type* information of the policy, and Δ the rules.

We treat an OWL DL Knowledge Base (KB) K as a set of statements in the so-called abstract syntax.[2] Each statement is either a *declaration* (see Table 1) or an *axiom* (see Table 2). Note that an abstract syntax "statement" such as $\texttt{Class}(A\ \texttt{partial}\ B)$ gives rise to multiple statements in the sense we intend (in this case, both a declaration $\texttt{Class}(A\ \ldots)$ and a subclass axiom $A\ \texttt{partial}\ B$). This should be clear from the tables. We let K_d be the set of declarations in K and K_a the set of axioms in K.

The translation of K is defined as

$$\begin{aligned}
\mathcal{T}(K) &= (\mathcal{T}_\Gamma(K), \mathcal{T}_\Delta(K)), \text{ where} \\
\mathcal{T}_\Gamma(K) &= \Gamma_{pre} \cup (\bigcup\{\mathcal{T}_\gamma(x) | x \in K_d\}) \\
\mathcal{T}_\Delta(K) &= \Delta_{pre} \cup (\bigcup\{\mathcal{T}_\delta(x) | x \in K_a\})
\end{aligned}$$

The translation functions for statements, $\mathcal{T}_\gamma$ and $\mathcal{T}_\delta$, are defined in Tables 1 and 2, respectively. These functions are *set*-valued. Unless otherwise noted, they denote a singleton set (i.e., the set with the one given formula), but in some cases multiple formulas are generated, as indicated by the use of indices i, j in the given formula, which range over the indices used in the first column.

Γ_{pre} and Δ_{pre} are a "prelude" that is always included in the translated policy:

$$\begin{aligned}
\Gamma_{pre} &= \{\texttt{Thing} : \texttt{Type}, \\
&\quad \texttt{str} : [\texttt{Data}] \to \texttt{Data},\ \texttt{ctor}(\texttt{str}),\ \texttt{String} : \texttt{Data} \to \texttt{Prop},\ \texttt{ind}(\texttt{String}), \\
&\quad \texttt{float} : \texttt{Data} \to \texttt{Data},\ \texttt{ctor}(\texttt{float}),\ \texttt{Float} : \texttt{Data} \to \texttt{Prop},\ \texttt{ind}(\texttt{Float})\} \\
\Delta_{pre} &= \{(\forall x : \texttt{Data})\texttt{Int}(x)\ \texttt{implies}\ \texttt{Rat}(x), \\
&\quad (\forall l : [\texttt{Data}])((\forall x : \texttt{Data}) x \in_l l\ \texttt{implies}\ \texttt{Int}(x))\ \texttt{iff}\ \texttt{String}(\texttt{str}(l)), \\
&\quad (\forall r : \texttt{Data})\texttt{Rat}(r)\ \texttt{implies}\ \texttt{Float}(\texttt{float}(r))\}
\end{aligned}$$

The tables show the $\mathcal{SHOIN}(\mathbf{D})$ formulas corresponding to the abstract syntax statements, where applicable (the OWL DL *declarations* have no corresponding $\mathcal{SHOIN}(\mathbf{D})$ formula). The translation functions make use of the auxiliary functions $\mathcal{T}_c$ for translating OWL concepts (Table 3), $\mathcal{T}_d$ for translating OWL data ranges (Table 5), $\mathcal{T}_r$ (Table 4) for translating OWL relations, and $\mathcal{T}_v$ for translating literals (Table 6).

2.1 Conjectures

(1) *Completeness.* If OWL DL KB $K \models_{DL} \phi$ then $\mathcal{T}(K) \models_{UPL} \mathcal{T}(\phi)$.

(2) *Soundness.* If $\mathcal{T}(K) \models_{UPL} \mathcal{T}(\phi)$ then $K \models_{DL} \phi$.

These should be easy to prove given that the semantics on the two sides is essentially the same.

[1] which corresponds directly to the Description Logic $\mathcal{SHOIN}(\mathbf{D})$ [HPSvH03]

[2] See http://www.w3.org/TR/owl-semantics/syntax.html

2.2 Notes

In the `Individual` statements in Tables 1 and 2, if there is no identifier o (i.e., it is a so-called anonymous individual), we give it a new unused name and treat it as a named individual.

We assume that all the OWL DL class and property names are also UPL propositional variables, and that all OWL DL individual names (including our named anonymous individuals) are UPL ordinary variables.

The translation works on OWL KBs, not on ontologies (files). Therefore, all statements in all ontologies loaded into the KB, and the import closure of all such policies, will be translated into one UPL policy. An alternative would be to treat imported ontologies as separate UPL policies, which are *composed* with the top-level policy, but we chose to avoid this additional complication for the time being.

All the ontologies in the OWL DL KB must obey all the restrictions for well-formed OWL DL ontologies.[3] In particular, all typed literal values must be well-typed.

In Tables 5 and 6, we use "xsd:" as a shorthand for "http://www.w3.org/2001/XMLSchema". As a future expansion, we could do something more interesting with certain XSD datatypes, for example, adding operations to `xsd:dateTime`. However, for the time being, we are no worse off than OWL in this regard, as OWL has no built-in support to do anything with these data types. In addition, we already allow operations on strings to be defined (in UPL), since strings are treated as lists.

Our approach shares some aspects with the one described in [LMKB04] in that they also translate OWL DL to a sorted first-order logic (in their case, CASL). Like them, we have two top-level sorts in the target language: `Thing` that contains all OWL individuals, and `Data` that contains all data values (e.g., integers, booleans, strings). One of the differences between our approach and theirs is that CASL has a more expressive sort system than UPL, which lets them translate some OWL class axioms into their sort system, whereas OWL classes are translated to ordinary predicates in our approach.

One noteworthy feature of our translation is the use of UPL's κ operator, which makes it straightforward to translate cardinality restrictions.

3 Translation of SWRL FOL and SWRL

The SWRL W3C submission[4] specifies an extension to the abstract syntax and semantics of OWL to handle Horn-like "rules". A subsequent submission[5] defines a further extension toward First-Order Logic.

Table 7 extends $\mathcal{T}_\delta$ to handle SWRL rules and SWRL FOL assertions (collectively known as SWRL *axioms*). The translation function makes use of the auxiliary functions $\mathcal{T}_a$ for translating SWRL axioms (Table 8), $\mathcal{T}_f$ and $\mathcal{T}_{var}$ for translating SWRL FOL formulae and variable declarations (Tables 9 and 10), and $\mathcal{T}_{io}$ and $\mathcal{T}_{do}$ for translating SWRL I-Objects and D-Objects (Tables 11 and 12).

We will make use of the functions FV_i and FV_v, which, when applied to a SWRL axiom or formula (see Table 9), returns the sequence of free *individual variables* or *data variables*, respectively, from the axiom/formula. The function $declareVars$ generates UPL variable declarations for all free variables in a SWRL axiom/formula, that is,

$$declareVars(\phi) \quad = \quad FV_i(\phi) : \texttt{Thing}, FV_v(\phi) : \texttt{Data}$$

We will also need the function AND, which is the UPL counterpart to the logical $\bigwedge$. For example,

$$AND_{1 \leq i \leq 3}(P_i(x)) \quad = \quad P_1(x) \texttt{ and } P_2(x) \texttt{ and } P_3(x)$$

We show only a subset of the SWRL "built-ins" (but we note that there was previously no formal specification

[3] See http://www.w3.org/TR/2004/REC-owl-ref-20040210/#OWLDL
[4] http://www.w3.org/Submission/SWRL/
[5] http://www.w3.org/Submission/SWRL-FOL/

for *any* of the built-ins). The translation for the rest of the built-ins can easily be added. Some require recursive functions, which would then be added to the prelude (see above).

References

[HPSvH03] I. Horrocks, P. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[LMKB04] Klaus Lüttich, Till Mossakowski, and Bernd Krieg-Brückner. Ontologies for the semantic web in Casl. In José Luiz Fiadeiro, Peter D. Mosses, and Fernando Orejas, editors, *WADT*, volume 3423 of *Lecture Notes in Computer Science*, pages 106–125. Springer, 2004.

Table 1: Translation of declarations. A, U, R, o, and D are URI references.

Abstract Syntax, Declarations	Translation, $\mathcal{T}_\gamma$
`Class`(A ...)	A : `Thing` $\rightarrow$ `Prop`
`DatatypeProperty`(U ... `Functional`)	U : `Thing` $\rightarrow$ `Data`
`DatatypeProperty`(U ...)	U : `Thing` $*$ `Data` $\rightarrow$ `Prop`
`ObjectProperty`(R ... `Functional`)	R : `Thing` $\rightarrow$ `Thing`
`ObjectProperty`(R ...)	R : `Thing` $*$ `Thing` $\rightarrow$ `Prop`
`Individual`(o ...)	o : `Thing`
`Datatype`(D)	D : `Data` $\rightarrow$ `Prop`

Table 2: Translation of axioms.

Abstract Syntax, Axioms (A)	DL Syntax	Translation, $\mathcal{T}_\delta(A)$
`Class`(A		
`partial` $C_1 \ \ldots \ C_n$	$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$(\forall x : \texttt{Thing})A(x)$ `implies` $\mathcal{T}_c(C_1, x)$ `and` $\ldots$ `and` $\mathcal{T}_c(C_n, x)$
`complete` $C_1 \ \ldots \ C_k$)	$A = C_1 \sqcap \ldots \sqcap C_k$	$(\forall x : \texttt{Thing})A(x)$ `iff` $\mathcal{T}_c(C_1, x)$ `and` $\ldots$ `and` $\mathcal{T}_c(C_k, x)$
`EnumeratedClass`($A \ o_1 \ \ldots \ o_n$)	$A = \{o_1, \ldots, o_n\}$	$(\forall x : \texttt{Thing})A(x)$ `iff` $x \in_s \{o_1\} \mid_s \ldots \mid_s \{o_n\}$
`SubClassof`($C_1 \ C_2$)	$C_1 \sqsubseteq C_2$	$(\forall x : \texttt{Thing})\mathcal{T}_c(C_1, x)$ `implies` $\mathcal{T}_c(C_2, x)$
`EquivalentClasses`($C_1 \ \ldots \ C_n$)	$C_i = C_j$	$(\forall x : \texttt{Thing})\mathcal{T}_c(C_i, x)$ `iff` $\mathcal{T}_c(C_j, x)$
`DisjointClasses`($C_1 \ \ldots \ C_n$)	$C_i \sqcap C_j = \bot, i \neq j$	$(\forall x : \texttt{Thing})\mathcal{T}_c(C_i, x)$ `iff not` $\mathcal{T}_c(C_j, x)$
`DatatypeProperty`(U		
`super`(U_1) $\ldots$ `super`(U_n)	$U \sqsubseteq U_i$	$(\forall x : \texttt{Thing}, y : \texttt{Data})\mathcal{T}_r(U, x, y)$ `implies` $\mathcal{T}_r(U_i, x, y)$
`domain`(C_1) $\ldots$ `domain`(C_k)	$\geqslant 1U \sqsubseteq C_i$	$(\forall x : \texttt{Thing}, y : \texttt{Data})\mathcal{T}_r(U, x, y)$ `implies` $\mathcal{T}_c(C_i, x)$
`range`(D_1) $\ldots$ `range`(D_l))	$\top \sqsubseteq \forall U.D_i$	$(\forall x : \texttt{Thing}, y : \texttt{Data})\mathcal{T}_r(U, x, y)$ `implies` $\mathcal{T}_d(D_i, y)$
`SubPropertyOf`($U_1 U_2$)	$U_1 \sqsubseteq U_2$	$(\forall x : \texttt{Thing}, y : \texttt{Data})\mathcal{T}_r(U_1, x, y)$ `implies` $\mathcal{T}_r(U_2, x, y)$
`EquivalentProperties`($U_1 \ \ldots \ U_n$)	$U_i = U_j$	$(\forall x : \texttt{Thing}, y : \texttt{Data})\mathcal{T}_r(U_i, x, y)$ `iff` $U_j(x, y)$
`ObjectProperty`(R		
`super`(R_1) $\ldots$ `super`(R_n)	$R \sqsubseteq R_i$	$(\forall x, y : \texttt{Thing})\mathcal{T}_r(R, x, y)$ `implies` $\mathcal{T}_r(R_i, x, y)$
`domain`(C_1) $\ldots$ `domain`(C_k)	$\geqslant 1R \sqsubseteq C_i$	$(\forall x, y : \texttt{Thing})\mathcal{T}_r(R, x, y)$ `implies` $\mathcal{T}_c(C_i, x)$
`range`(C_k) $\ldots$ `range`(C_l)	$\top \sqsubseteq \forall R.C_i$	$(\forall x, y : \texttt{Thing})\mathcal{T}_r(R, x, y)$ `implies` $\mathcal{T}_c(C_i, y)$
`inverseOf`(R_0)	$R =^- R_0$	$(\forall x, y : \texttt{Thing})\mathcal{T}_r(R, x, y)$ `iff` $\mathcal{T}_r(R_0, y, x)$
`Symmetric`	$R =^- R$	$(\forall x, y : \texttt{Thing})\mathcal{T}_r(R, x, y)$ `iff` $\mathcal{T}_r(R, y, x)$
`Transitive`	$Tr(R)$	$(\forall x, y, z : \texttt{Thing})\mathcal{T}_r(R, x, z)$ `and` $\mathcal{T}_r(R, z, y)$ `implies` $\mathcal{T}_r(R, x, y)$
`InverseFunctional`)	$\top \sqsubseteqslant 1R^-$	$(\forall x : \texttt{Thing})(\kappa y : \texttt{Thing})\mathcal{T}_r(R, y, x) \leq 1$
`SubPropertyOf`($R_1 R_2$)	$R_1 \sqsubseteq R_2$	$(\forall x, y : \texttt{Thing})\mathcal{T}_r(R_1, x, y)$ `implies` $\mathcal{T}_r(R_2, x, y)$
`EquivalentProperties`($R_1 \ \ldots \ R_n$)	$R_i = R_j$	$(\forall x, y : \texttt{Thing})\mathcal{T}_r(R_i, x, y)$ `iff` $\mathcal{T}_r(R_j, x, y)$
`Individual`(o		
`type`(C_1) $\ldots$ `type`(C_n)	$o \in C_i$	$\mathcal{T}_c(C_i, o)$
`value`($R_1 o_1$) $\ldots$ `value`($R_k o_k$)	$\langle o, o_i \rangle \in R_i$	$\mathcal{T}_r(R_i, o, o_i)$
`value`($U_1 v_1$) $\ldots$ `value`($U_l v_l$))	$\langle o, v_i \rangle \in U_i$	$\mathcal{T}_r(U_i, o, \mathcal{T}_v(v_i))$
`SameIndividual`($o_1 \ \ldots \ o_n$)	$o_i = o_j$	$o_i = o_j$
`DifferentIndividuals`($o_1 \ \ldots \ o_n$)	$o_i \neq o_j, i \neq j$	$o_i \not\equiv o_j$

Table 3: Translation of concepts.

Abstract Syntax, Descriptions (C)	DL Syntax	Translation, $\mathcal{T}_c(C, x)$
A (URI reference)	A	$A(x)$
`owl:Thing`	$\top$	`True`
`owl:Nothing`	$\bot$	`False`
`intersectionOf`$(C_1\ C_2\ \dots)$	$C_1 \sqcap C_2 \sqcap \dots$	$\mathcal{T}_c(C_1, x)$ `and` $\mathcal{T}_c(C_2, x)$ `and` $\dots$
`unionOf`$(C_1\ C_2\ \dots)$	$C_1 \sqcup C_2 \sqcup \dots$	$\mathcal{T}_c(C_1, x)$ `or` $\mathcal{T}_c(C_2, x)$ `or` $\dots$
`complementOf`(C)	$\neg C$	`not` $\mathcal{T}_c(C, x)$
`oneOf`$(o_1\ o_2\ \dots)$	$\{o_1, o_2, \dots\}$	$x \in_s \{o_1\} \mid_s \{o_2\} \mid_s \dots$
If `ObjectProperty`$(R\ \dots) \in K$		
`restriction`$(R$ `someValuesFrom`$(C))$	$\exists R.C$	$(\exists y : \texttt{Thing})\mathcal{T}_r(R, x, y)$ `and` $\mathcal{T}_c(C, y)$
`restriction`$(R$ `allValuesFrom`$(C))$	$\forall R.C$	$(\forall y : \texttt{Thing})\mathcal{T}_r(R, x, y)$ `implies` $\mathcal{T}_c(C, y)$
`restriction`$(R$ `hasValue`$(o))$	$R : o$	$\mathcal{T}_r(R, x, o)$
`restriction`$(R$ `minCardinality`$(n))$	$\geqslant nR$	$(\kappa y : \texttt{Thing})\mathcal{T}_r(R, x, y) \geq n$
`restriction`$(R$ `maxCardinality`$(n))$	$\leqslant nR$	$(\kappa y : \texttt{Thing})\mathcal{T}_r(R, x, y) \leq n$
`restriction`$(R$ `cardinality`$(n))$	$= nR$	$(\kappa y : \texttt{Thing})\mathcal{T}_r(R, x, y) = n$
If `DatatypeProperty`$(U\ \dots) \in K$		
`restriction`$(U$ `someValuesFrom`$(D))$	$\exists U.D$	$(\exists y : \texttt{Data})\mathcal{T}_r(U, x, y)$ `and` $\mathcal{T}_d(D, y)$
`restriction`$(U$ `allValuesFrom`$(D))$	$\forall U.D$	$(\forall y : \texttt{Data})\mathcal{T}_r(U, x, y)$ `implies` $\mathcal{T}_d(D, y)$
`restriction`$(U$ `hasValue`$(v))$	$U : v$	$\mathcal{T}_r(U, x, \mathcal{T}_v(v))$
`restriction`$(U$ `minCardinality`$(n))$	$\geqslant nU$	$(\kappa y : \texttt{Data})\mathcal{T}_r(U, x, y) \geq n$
`restriction`$(U$ `maxCardinality`$(n))$	$\leqslant nU$	$(\kappa y : \texttt{Data})\mathcal{T}_r(U, x, y) \leq n$
`restriction`$(U$ `cardinality`$(n))$	$= nU$	$(\kappa y : \texttt{Thing})\mathcal{T}_r(U, x, y) = n$

Table 4: Translation of relations.

Abstract Syntax, Relations (R)	DL Syntax	Translation, $\mathcal{T}_r(R, x, y)$
If `ObjectProperty`$(R\ \dots$ `Functional`$) \in K$		
R (a URI reference)	R	$R\ x = y$
Else if `ObjectProperty`$(R\ \dots) \in K$		
R (a URI reference)	R	$R(x, y)$
If `DatatypeProperty`$(U\ \dots$ `Functional`$) \in K$		
U (a URI reference)	U	$U\ x = y$
Else if `DatatypeProperty`$(U\ \dots) \in K$		
U (a URI reference)	U	$U(x, y)$

Table 5: Translation of data ranges.

Abstract Syntax, Data Ranges (D)	DL Syntax	Translation, $\mathcal{T}_d(D, x)$
`oneOf`$(v_1\ v_2\ \ldots)$	$\{v_1, v_2, \ldots\}$	$x \in_s \{\mathcal{T}_v(v_1)\} \mid_s \{\mathcal{T}_v(v_2)\} \mid_s \ldots$
`xsd:boolean`		`Bool`(x)
`xsd:decimal`		`Rat`(x)

`xsd:integer`
`xsd:long`
`xsd:int`
`xsd:short`
		`Int`(x)

`xsd:nonPositiveInteger`		`Int`(x) and $x \leq 0$
`xsd:negativeInteger`		`Int`(x) and $x < 0$

`xsd:nonNegativeInteger`
`xsd:unsignedLong`
`xsd:unsignedInt`
`xsd:unsignedShort`
`xsd:unsignedByte`
`xsd:hexBinary`
		`Int`(x) and $x \geq 0$

`xsd:positiveInteger`		`Int`(x) and $x > 0$
`xsd:float` `xsd:double`		`Float`(x)
`rdf:XMLLiteral` any other builtin xsd datatype		`String`(x)
any user-defined datatype		`Data`(x)

Table 6: Translation of literals.

Abstract Syntax, Literals (v)	Translation, $\mathcal{T}_v(v)$
v`^^xsd:boolean`	the boolean represented by v
v`^^xsd:decimal`	the rational represented by v
v`^^xsd:integer` v`^^xsd:long` v`^^xsd:int` v`^^xsd:short` v`^^xsd:nonPositiveInteger` v`^^xsd:negativeInteger` v`^^xsd:nonNegativeInteger` v`^^xsd:unsignedLong` v`^^xsd:unsignedInt` v`^^xsd:unsignedShort` v`^^xsd:unsignedByte` v`^^xsd:hexBinary` v`^^xsd:positiveInteger`	the integer represented by v
v`^^xsd:float` v`^^xsd:double`	`float`(v_r) where v_r is the rational corresponding to v
v (untyped literal) or v`^^`t where v consists of the characters $c_1, \ldots, c_n$ and t is any supported XSD datatype not mentioned above or `rdf:XMLLiteral`	`str`$([c2u(c_1)] \mid_l \ldots \mid_l [c2u(c_n)])$ where $c2u$ is the function from characters to Unicode character codes (integers).

Table 7: Translation of SWRL Rules and SWRL FOL Assertions.

Abstract Syntax, Rule (R)	Translation, $\mathcal{T}_\delta(R)$

`Implies(`
  `Antecedent`$(A_1, \ldots, A_n)$
  `Consequent`$(B_1, \ldots, B_k)$)
	$(\forall\ declareVars(R))$
	$\mathcal{T}_a(A_1)$ `and` $\ldots$ `and` $\mathcal{T}_a(A_n)$ `implies`
	$\mathcal{T}_a(B_1)$ `and` $\ldots$ `and` $\mathcal{T}_a(B_n)$

`Assertion`(F)	$\mathcal{T}_f(F)$

Table 8: Translation of SWRL atoms.

Abstract Syntax, Atom (A)	Translation, $\mathcal{T}_a(A)$
$C(i)$	$\mathcal{T}_c(C, \mathcal{T}_{io}(i))$
$D(v)$	$\mathcal{T}_d(D, \mathcal{T}_{do}(v))$
$R(i_1, i_2)$	$\mathcal{T}_r(R, \mathcal{T}_{io}(i_1), \mathcal{T}_{io}(i_2))$
$U(v_1, v_2)$	$\mathcal{T}_r(U, \mathcal{T}_{do}(v_1), \mathcal{T}_{do}(v_2))$
`sameAs`$(i_1\ i_2)$	$\mathcal{T}_{io}(i_1) = \mathcal{T}_{io}(i_2)$
`differentFrom`$(i_1\ i_2)$	$\mathcal{T}_{io}(i_1) \neq \mathcal{T}_{io}(i_2)$
`builtIn(swrlb:equal` $\mathtt{v_1}\ \ldots\ \mathtt{v_n}$`)`	$\mathcal{T}_{do}(v_1) = \mathcal{T}_{do}(v_2)$ `and` $\ldots$ `and` $\mathcal{T}_{do}(v_{n-1}) = \mathcal{T}_{do}(v_n)$
`builtIn(swrlb:notEqual` $\mathtt{v_1}\ \ldots\ \mathtt{v_n}$`)`	$AND_{1 \leq i,j \leq n, i \neq j}(\mathcal{T}_{do}(v_i) \not\equiv \mathcal{T}_{do}(v_j))$
`builtIn(swrlb:lessThan` $\mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_1) < \mathcal{T}_{do}(v_2)$
`builtIn(swrlb:lessThanOrEqual` $\mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_1) \leq \mathcal{T}_{do}(v_2)$
`builtIn(swrlb:greaterThan` $\mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_1) > \mathcal{T}_{do}(v_2)$
`builtIn(swrlb:greaterThanOrEqual` $\mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_1) \geq \mathcal{T}_{do}(v_2)$
`builtIn(swrlb:add` $\mathtt{v_r}\ \mathtt{v_1}\ \ldots\ \mathtt{v_n}$`)`	$\mathcal{T}_{do}(v_r) = \mathcal{T}_{do}(v_1) + \ldots + \mathcal{T}_{do}(v_n)$
`builtIn(swrlb:subtract` $\mathtt{v_r}\ \mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_r) = \mathcal{T}_{do}(v_1) - \mathcal{T}_{do}(v_2)$
`builtIn(swrlb:multiply` $\mathtt{v_r}\ \mathtt{v_1}\ \ldots\ \mathtt{v_n}$`)`	$\mathcal{T}_{do}(v_r) = \mathcal{T}_{do}(v_1) * \ldots * \mathcal{T}_{do}(v_n)$
`builtIn(swrlb:divide` $\mathtt{v_r}\ \mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_r) = \mathcal{T}_{do}(v_1)/\mathcal{T}_{do}(v_2)$
`builtIn(swrlb:integerDivide` $\mathtt{v_r}\ \mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_r) = \mathcal{T}_{do}(v_1)$ `div` $\mathcal{T}_{do}(v_2)$
`builtIn(swrlb:mod` $\mathtt{v_r}\ \mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_r) = \mathcal{T}_{do}(v_1)$ `mod` $\mathcal{T}_{do}(v_2)$
`builtIn(swrlb:stringConcat` $\mathtt{v_r}\ \mathtt{v_1}\ \ldots\ \mathtt{v_n}$`)`	$\mathcal{T}_{do}(v_r) = \mathcal{T}_{do}(v_1) \mid_l \ldots \mid_l \mathcal{T}_{do}(v_n)$
`builtIn(swrlb:listConcat` $\mathtt{v_r}\ \mathtt{v_1}\ \ldots\ \mathtt{v_n}$`)`	$\mathcal{T}_{do}(v_r) = \mathcal{T}_{do}(v_1) \mid_l \ldots \mid_l \mathcal{T}_{do}(v_n)$
`builtIn(swrlb:member` $\mathtt{v_1}\ \mathtt{v_2}$`)`	$\mathcal{T}_{do}(v_1) \in_l \mathcal{T}_{do}(v_2)$
`builtIn(swrlb:empty` $\mathtt{v}$`)`	$\mathcal{T}_{do}(v) = [\,]$

Table 9: Translation of SWRL FOL formulae.

Abstract Syntax, Formula (F)	Translation, $\mathcal{T}_f(F)$
A	$\mathcal{T}_a(A)$
`and`$(F_1\ \ldots\ F_n)$	$\mathcal{T}_f(F_1)$ `and` $\ldots$ `and` $\mathcal{T}_f(F_n)$
`or`$(F_1\ \ldots\ F_n)$	$\mathcal{T}_f(F_1)$ `or` $\ldots$ `or` $\mathcal{T}_f(F_n)$
`neg`(F)	`not` $\mathcal{T}_f(F)$
`implies`$(F_1\ F_2)$	$\mathcal{T}_f(F_1)$ `implies` $\mathcal{T}_f(F_2)$
`equivalent`$(F_1\ F_2)$	$\mathcal{T}_f(F_1)$ `iff` $\mathcal{T}_f(F_2)$
`forall`$(V_1\ \ldots\ V_n\ F)$	$(\forall\ declareVars(F))\mathcal{T}_{var}(V_1)$ `and` $\ldots$ `and` $\mathcal{T}_{var}(V_n)$ `implies` $\mathcal{T}_f(F)$
`exists`$(V_1\ \ldots\ V_n\ F)$	$(\exists\ declareVars(F))\mathcal{T}_{var}(V_1)$ `and` $\ldots$ `and` $\mathcal{T}_{var}(V_n)$ `and` $\mathcal{T}_f(F)$

Table 10: Translation of SWRL FOL Variable Declarations.

Abstract Syntax, Variable Declaration (V)	Translation, $\mathcal{T}_{var}(V)$
`I-variable`$(i\ C)$ (i a URI reference)	$\mathcal{T}_c(C, i)$
`D-variable`$(v\ D)$ (v a URI reference)	$\mathcal{T}_d(D, v)$

Table 11: Translation of SWRL I-Objects.

Abstract Syntax, I-Objects (i)	Translation, $\mathcal{T}_{io}(i)$
`I-variable`(i) (i a URI reference)	i
o (o a URI reference)	o

Table 12: Translation of SWRL D-Objects.

Abstract Syntax, D-Objects (d)	Translation, $\mathcal{T}_{do}(d)$
`D-variable`(d) (d a URI reference)	d
v	$\mathcal{T}_v(v)$