From OWL 2 to DLGP: the ER Profile
Technical Report

Jean-François Baget, Alain Gutierrez, Michel Leclère, Marie-Laure Mugnier,
Swan Rocher, and Clément Sipieter

Inria, CNRS and University of Montpellier
France

1 Introduction

We introduce here the ER (for Existential Rule) profile of OWL 2, for which all axioms can be translated into *dlgp*[1] statements. We point out that all axioms that can be written in existing profiles of OWL 2 (namely EL, QL and RL) are axioms of ER.

For the sake of simplicity, we do not discuss here datatypes nor literals. Axioms used for datatypes and literals always correspond to a similar axiom used for classes and individuals (for instance `DataIntersectionOf` corresponds to `ObjectIntersectionOf`). They are thus processed similarly in our translation.

2 Preliminary Notions

Basic objects in an OWL 2 ontology are *entities*, such as *classes*, *properties* and *individuals*. These entities are identified by IRIs. We associate an OWL 2 individual `i` with the logical constant i, an OWL 2 class `C` with the unary predicate C, and an OWL 2 property `p` with the binary predicate p.[2]

Entities are used to build *expressions*, such as *class expressions* or *property expressions*. We present these expressions both in OWL 2 functional notation, such as `ObjetIntersectionOf(A, ObjectComplementOf(B))`, and in their DL notation such as $A \sqcap \neg B$; they both identify the class whose elements are in A and not in B. For every class expression C, we can build a FOL formula $\Phi_C(x)$ whose only free variable is x, expressing that "x is an element of the class C". For instance, $\Phi_{A \sqcap \neg B}(x) = A(x) \wedge \neg B(x)$. In the same way, for every property expression p, we can build a FOL formula $\Phi_P(x, y)$ whose only free variables are x and y, expressing that "the relation p holds between the subject x and the object y".

[1] `https://graphik-team.github.io/graal/dl/datalog+_v2.0_en.pdf`

[2] We already discuss here the particular case of two specific classes, `Thing` and `Nothing` (respectively written $\top$ and $\bot$ in DL). `Thing` is the universal class that contains everything and `Nothing` is the empty class. They are used as any other class in our framework, though their particular semantics is expressed in *dlgp* by the two following *dlgp* statements that must be present in every *dlgp* knowledge base translating an OWL 2 ontology: the *dlgp* constraint `! :- Nothing(X)`; and the *dlgp* annotation `@top Thing` that declares that the universal class in the knowledge base is named `Thing`.

An OWL 2 ontology is a set of *axioms*, built from expressions (we do not discuss here *annotations*, which have no logical translation). The axiom `SubclassOf(A, B)` means that all elements of A are also elements of B. It is written $A \sqsubseteq B$ in DL notation. This axiom is translated into a FOL formula (without free variable) $\forall x\,(A(x) \rightarrow B(x))$. Almost all OWL 2 axioms can be translated into formulas of the form $\forall \vec{x}(\mathcal{B}(\vec{x}) \rightarrow \mathcal{H}(\vec{x}))$ where $\mathcal{B}(\vec{x})$ and $\mathcal{H}(\vec{x})$ are FOL formulas whose only free variable is x. These formulas cannot always be translated into *dlgp*, as shown in Example 1.

Example 1. The axiom $C \sqsubseteq A \sqcap \neg B$ is translated by the formula $\forall x\,(C(x) \rightarrow A(x) \wedge \neg B(x)$. It is equivalent to the conjunction of the two formulas $\forall x\,(C(x) \rightarrow A(x))$ and $\forall x\,(C(x) \rightarrow \neg B(x))$. The first is expressed by the *dlgp* rule `A(X) :- C(X)` and the second by the *dlgp* constraint `! :- B(X), C(X).` In contrast, the axiom $A \sqcap \neg B \sqsubseteq C$ cannot be translated into *dlgp*.

The ER (for existential rules) profile of OWL 2 is obtained by putting syntactic restrictions on OWL 2 expressions and axioms, in order to ensure that all axioms have an equivalent translation in *dlgp*. This profile defines different kinds of class expressions, according to the position they can fill in a formula of the form $\forall \vec{x}(\mathcal{B}(\vec{x}) \rightarrow \mathcal{H}(\vec{x}))$. *EquivClass* expressions can appear in both sides of such an implication, as will be discussed in Sect. 3. *SubClass* expressions can only appear in the left side (Sect. 4), while *SuperClass* expressions can only appear in the right side (Sect. 5). We show in Sect. 6 that any OWL 2 axiom can either be easily translated into *dlgp* or is equivalent to a formula of the form $\forall \vec{x}(\mathcal{B}(\vec{x}) \rightarrow \mathcal{H}(\vec{x}))$, that can be translated when it complies with the restrictions of the ER profile.

In this paper, all axioms and expression constructors will be presented according to the format given in Tab. 1.

Type of axiom or expression		
Name of axiom or expression		
`Axiom or expression in OWL 2 functional syntax`	DL syntax	Logical translation
Optional comments.		

Table 1. General format of tables

3 EquivClass expressions

A FOL formula $\mathcal{F}(\vec{x})$ is said to be *conjunctive* when it is in the form $\exists \vec{z}\,(C_1[\vec{x},\vec{z}] \wedge \ldots \wedge C_p[\vec{x},\vec{z}])$ where the $C_i[\vec{x},\vec{z}]$ are (positive) atoms whose variables are in $\vec{x} \cup \vec{z}$.[3]

Property 1. For every property expression p, $\Phi_p(x, y)$ is a conjunctive formula. See Tab. 2.

[3] Moreover, we always *simplify* such a conjunctive formula: it is equivalent to $\texttt{Nothing}(x)$ if one of its atoms is some $\texttt{Nothing}(y)$, and we can remove all atoms of the form $\texttt{Thing}(y)$ without changing the semantics (unless the formula is restricted to a single atom $\texttt{Thing}(x)$).

Object Property Expressions		
Object Property		
`p`	p	$p(x,y)$
Inverse Object Property		
`ObjectInverseOf(p)`	p^-	$p(y,x)$
Property Expression Chain		
`ObjectPropertyChain`$(p_1,\ldots,p_k)$	$p_1 \cdot \ldots \cdot p_k$	$\exists z_1 \ldots \exists z_{k-1}(\Phi_{p_1}(x,z_1) \wedge \ldots \wedge \Phi_{p_k}(z_{k-1},y))$
Note that the arguments of a property expression chain are always object property expressions.		

Table 2. Property expressions in OWL 2.

Proof. All OWL 2 property expression constructors are listed in Tab. 2. The property is immediate.

In the ER profile, an *EquivClass* expression is a class expression built, without any other restriction, from the constructors listed in Tab. 3.

EquivClass expressions		
Class		
`C`	C	$C(x)$
Intersection of Class Expressions		
`ObjectIntersectionOf`$(C_1,\ldots,C_k)$	$C_1 \sqcap \ldots \sqcap C_k$	$\Phi_{C_1}(x) \wedge \ldots \wedge \Phi_{C_k}(x)$
Existential Quantification		
`ObjectSomeValuesFrom`(p,C)	$\exists p \cdot C$	$\exists y\,(\Phi_p(x,y) \wedge \Phi_C(y))$
Individual Value Restriction		
`ObjectHasValue`(p,i)	$\exists p \cdot \{i\}$	$\Phi_p(x,i)$
Self-Restriction		
`ObjectHasSelf`(p)	$\exists p \cdot \mathrm{Self}$	$\Phi_p(x,x)$
Minimum Cardinality - Restricted to n = 0 or 1		
`ObjectMinCardinality`$(0,p,C)$	$\geq 0pC$	`Thing`(x)
`ObjectMinCardinality`$(1,p,C)$	$\geq 1pC$	$\exists y\,(\Phi_p(x,y) \wedge \Phi_C(y))$
Enumeration of Individuals - Restricted to n = 1		
`ObjectOneOf`(i)	$\{i\}$	$x = i$

Table 3. EquivClass expressions constructors

Property 2. For every *EquivClass* expression C, $\Phi_C(x)$ is equivalent to a conjunctive formula.

Proof. Consider the formula $\Phi_C(x)$ built from the constructors in Tab. 2 and 3. By putting it into prenex form, then simplifying it, we obtain a conjunctive formula.

The following property is the basis of our transformation from OWL 2 to *dlgp*.

Property 3. Every formula of the form $\forall \vec{x}(\mathcal{B}(\vec{x}) \rightarrow \mathcal{H}(\vec{x}))$ where $\mathcal{B}(\vec{x})$ and $\mathcal{H}(\vec{x})$ are conjunctive can be translated into an equivalent *dlgp* rule.

Proof. Let $\mathcal{B}(\vec{x}) = \exists \vec{y}\,(b_1[\vec{x},\vec{y}] \wedge \ldots \wedge b_k[\vec{x},\vec{y}])$ and $\mathcal{H}(\vec{x}) = \exists \vec{z}\,(h_1[\vec{x},\vec{z}] \wedge \ldots \wedge h_q[\vec{x},\vec{z}])$. Up to a variable renaming, we can consider that $\vec{y} \cap \vec{z} = \emptyset$. Then $\forall \vec{x}\,(\mathcal{B}(\vec{x}) \rightarrow \mathcal{H}(\vec{x}))$ is equivalent to the existential rule $\forall \vec{x} \forall \vec{y}((b_1[\vec{x},\vec{y}] \wedge \ldots \wedge b_k[\vec{x},\vec{y}]) \rightarrow \exists \vec{z}(h_1[\vec{x},\vec{z}] \wedge \ldots \wedge$

$h_q[\vec{x}, \vec{z}]))$, which can thus be translated into the *dlgp* rule $h_1[\vec{X}, \vec{Z}], \ldots, h_q[\vec{X}, \vec{Z}]$:- $b_1[\vec{X}, \vec{Y}], \ldots, b_k[\vec{X}, \vec{Y}]$

Example 2. The class expression $\exists p \cdot (\exists q \cdot C)$ is translated into FOL by $\Phi_{\exists p \cdot (\exists q \cdot C)}(x) = \exists y_1(p(x, y_1) \land (\exists y_2(q(y_1, y_2) \land C(y_2))))$. By putting it in prenex form, we obtain the conjunctive formula $\exists y_1 \exists y_2(p(x, y_1) \land q(y_1, y_2) \land C(y_2))$. Thus the axiom $D \sqsubseteq \exists p \cdot (\exists q \cdot C)$ is translated by the *dlgp* rule: `p(X, Y1), q(Y1, Y2), C(Y2) :- D(X).`

As a final remark on the translation of implications of conjunctive formulas, we point out that formulas of the form $\forall x(\mathcal{B}(x) \rightarrow \texttt{Thing}(x))$ or $\forall x(\texttt{Nothing}(x) \rightarrow \mathcal{H}(x))$ do not bring any information, thus do not need to be translated; that formulas of the form $\forall x(\mathcal{B}(x) \rightarrow \texttt{Nothing}(x))$ can be directly translated into a *dlgp* constraint; and that formulas of the form $\forall x(x = a \rightarrow \mathcal{B}(x))$ can be directly translated into a *dlgp* fact.

Example 3. The axiom $A \sqsubseteq \exists p \cdot \bot$ is translated by the FOL formula $\forall x(A(x) \rightarrow (\exists y(p(x, y) \land \texttt{Nothing}(y))))$, which can be simplified in $\forall x(A(x) \rightarrow \texttt{Nothing}(x))$, and thus can be expressed by the *dlgp* constraint `! :- A(X)`

The axiom $\{a\} \sqsubseteq \exists p \cdot C$ is translated by the FOL formula $\forall x\,((x = a) \rightarrow \exists y(p(x, y) \land C(y))$ and thus can be expressed by the *dlgp* fact: `p(a, Y), C(Y).`

4 SubClass expressions

A FOL formula $\mathcal{F}(\vec{x})$ is said to be *disjunctive* when it is a disjunction $\mathcal{F}_1(\vec{x}) \lor \ldots \lor \mathcal{F}_k(\vec{x})$ of conjunctive formulas. In that case, we say that the disjunction is of size k.[4]

In the ER profile, a *SubClass* expression is a class expression built, without any other restriction, from the constructors listed in Tab. 4.

EquivClass expressions		
All EquivClass expressions constructors: Atomic class expressions (including `Thing` and `Nothing`), `ObjectIntersectionOf`, `ObjectSomeValuesFrom`, `ObjectHasValue`, `ObjectHasSelf`, `ObjectMinCardinality` (restricted to $n = 0$ or 1), `ObjectOneOf` (restricted to $n = 1$).		
SubClass expressions		
Union of class expressions		
`ObjectUnionOf`$(C_1, \ldots, C_k)$	$C_1 \sqcup \ldots \sqcup C_k$	$\Phi_{C_1}(x) \lor \ldots \lor \Phi_{C_k}(x)$
Enumeration of individuals (unrestricted)		
`ObjectOneOf`$(i_1, \ldots, i_k)$	$\{i_1, \ldots, i_k\}$	$x = i_1 \lor \ldots \lor x = i_k$

Table 4. SubClass expressions constructors.

Property 4. If C is a *SubClass* expression, $\Phi_C(x)$ is equivalent to a disjunctive formula.

[4] We always *simplify* a disjunctive formula: it is equivalent to $\texttt{Thing}(x)$ if one of its conjunctive formulas is $\texttt{Thing}(x)$, and we can remove all conjunctive formulas of the form $\texttt{Nothing}(x)$ without changing the semantics (unless the formula is restricted to a single conjunctive formula $\texttt{Nothing}(x)$).

Proof. Consider the formula $\Phi_C(x)$ built from the constructors in Tab. 2 and 4. By putting it into prenex form, we obtain a formula whose atoms are connected only by disjunctions and conjunctions. By a sequence of transformations using distributivity, we obtain a disjunctive formula, that we can finally simplify.

Example 4. The SubClass expression $(A \sqcup B) \sqcap \exists p \cdot (A \sqcup B)$ is translated by the FOL formula $(A(x) \vee B(x)) \wedge \exists y(p(x,y) \wedge (A(y) \vee B(y)))$. It is equivalent to the disjunctive formula $\mathcal{F}_{AA}(x) \vee \mathcal{F}_{AB}(x) \vee \mathcal{F}_{BA}(x) \vee \mathcal{F}_{BB}(x)$ where $\mathcal{F}_{AA}(x) = \exists y(A(x) \wedge p(x,y) \wedge A(y))$, $\mathcal{F}_{AB}(x) = \exists y(A(x) \wedge p(x,y) \wedge B(y))$, $\mathcal{F}_{BA}(x) = \exists y(B(x) \wedge p(x,y) \wedge A(y))$ and $\mathcal{F}_{BB}(x) = \exists y(B(x) \wedge p(x,y) \wedge B(y))$.

Note that putting the formula translating a SubClass expression into its disjunctive form can be exponential in the size of the initial formula.

Property 5. Every formula of the form $\forall \vec{x}(\mathcal{B}(\vec{x}) \rightarrow \mathcal{H}(\vec{x}))$, where $\mathcal{B}(\vec{x})$ is a disjunctive formula of size k and $\mathcal{H}(\vec{x})$ is a conjunctive formula, can be translated into an equivalent conjunction of k *dlgp* rules.

Proof. See that a formula of form $\forall \vec{x}((\mathcal{B}_1(\vec{x}) \vee \ldots \vee (\mathcal{B}_k(\vec{x})) \rightarrow \mathcal{H}(\vec{x}))$ is equivalent to the conjunction of the k formulas, for $1 \leq i \leq k$, $\forall \vec{x}(\mathcal{B}_i(\vec{x}) \rightarrow \mathcal{H}(\vec{x}))$, where $\mathcal{B}_i(\vec{x})$ and $\mathcal{H}(\vec{x})$ are conjunctive formulas. It remains to conclude with property 3.

Example 5. The axiom $(A \sqcup B) \sqcap \exists p \cdot (A \sqcup B) \sqsubseteq \exists q \cdot \top$ is translated by the four following *dlgp* rules: `q(X, Z) :- A(X), p(X, Y), A(Y)` and `q(X, Z) :- A(X), p(X, Y), B(Y)` and `q(X, Z):- B(X), p(X, Y), A(Y)` and `q(X, Z) :- B(X), p(X, Y), B(Y).`

5 SuperClass expressions

Contrary to what happens with EquivClass and SubClass expressions, *all* OWL 2 constructors can appear in ER SuperClass expressions. Hence, these expressions can also use, in addition to the constructors already presented, the constructors listed in Tab. 5. However, we impose syntactic restrictions on the possible interactions between these constructors.

Definition 1. SuperClass *expressions are defined inductively. A SuperClass expression is either an EquivClass expression; the intersection $C_1 \sqcap \ldots \sqcap C_k$ of SuperClass expressions C_i; the complement $\neg C$ of a SubClass expression C; the universal restriction $\forall p \cdot C$ of a SuperClass expression C; or the maximum cardinality $\leq n\, p\, C$ of a SubClass expression C, when n is restricted to 0 or 1.*

Property 6. A formula $\forall x\, (\Phi_B(x) \rightarrow \Phi_H(x))$, where B is a SubClass expression and H is a SuperClass expression, is equivalent to a conjunction of formulas of the form $\forall x\, (\mathcal{B}(x) \rightarrow \mathcal{H}(x))$, where $\mathcal{B}(x)$ is disjunctive and $\mathcal{H}(x)$ is conjunctive.

Complement of Class Expressions		
`ObjectComplementOf`(C)	$\neg C$	$\neg \Phi_C(x)$
Is a SuperClass expression when C is a SubClass expression		
Universal Quantification		
`ObjectAllValuesFrom`(p, C)	$\forall p \cdot C$	$\forall y\, (\Phi_p(x, y) \rightarrow \Phi_C(y))$
Is a SuperClass expression when C is a SuperClass expression		
Maximum Cardinality		
`ObjectMaxCardinality`(n, p, C)	$\leq npC$	$\forall y_1 \ldots \forall y_{n+1}((\Phi_p(x, y_1) \wedge \Phi_C(y_1) \wedge \ldots \wedge \Phi_p(x, y_{n+1}) \wedge \Phi_C(y_{n+1})) \rightarrow \vee_{1 \leq i < j \leq n+1} y_i = y_j))$
Only used when n is restricted to 0 or 1, is a SuperClass expression when C is a SubClass expression.		
Exact Cardinality		
`ObjectExactCardinality`(n, p, C)	$= npC$	Macro for `ObjectMinCardinality` and `ObjectMaxCardinality`.

Table 5. List of all other (non datatype) OWL 2 constructors.

Proof. We show that property inductively on the SuperClass expression H.

If H is an *EquivClass* expression, then the property is immediate.

If $H = H_1 \sqcap \ldots \sqcap H_k$, then our formula is equivalent to the conjunction of formulas $\forall x\, (\Phi_B(x) \rightarrow \Phi_{H_i}(x))$, where the H_i are SuperClass expressions.

If $H = \neg H'$, then our formula is equivalent to $\forall x\, (\Phi_B(x) \wedge \Phi_{H'}(x) \rightarrow$ `Nothing`$(x))$. Since both B and H' are SubClass expressions, the conjunction of $\Phi_B(x)$ and $\Phi_{H'}(x)$ is equivalent to a disjunctive formula.

If $H = \forall p \cdot H'$, then our formula is equivalent to $\forall y\, (\exists x\, (\Phi_B(x) \wedge \Phi_p(x, y)) \rightarrow \Phi_{H'}(y))$. Since $\Phi_B(x)$ is disjunctive, its conjunction with $\exists y\; p(x, y)$ can also be put in disjunctive form, and $\Phi_{H'}(y)$ is a SuperClass expression.

If $H = \leq 0\, p\, H'$, then our formula is equivalent to $\forall x\, (\exists y\, (\Phi_B(x) \wedge \Phi_p(x, y) \wedge \Phi_{H'}(y)) \rightarrow$ `Nothing`$(x))$. Since both B and H' are SubClass expressions, the formula $\exists y\, (\Phi_B(x) \wedge \Phi_p(x, y) \wedge \Phi_{H'}(y))$ is equivalent to a disjunctive formula.

If $H = \leq 1\, p\, H'$, then our formula is equivalent to $\forall x\, (\exists y_1 \exists y_2\, (\Phi_B(x) \wedge \Phi_p(x, y_1) \wedge \Phi_{H'}(y_1) \wedge \Phi_p(x, y_2) \wedge \Phi_{H'}(y_2)) \rightarrow y_1 = y_2)$. Since both B and H' are SubClass expressions, the formula $\exists y_1 \exists y_2\, (\Phi_B(x) \wedge \Phi_p(x, y_1) \wedge \Phi_{H'}(y_1) \wedge \Phi_p(x, y_2) \wedge \Phi_{H'}(y_2))$ is equivalent to a disjunctive formula.

Example 6. Let $\{a\} \sqcup \exists p \cdot A \sqsubseteq (\exists q \cdot B) \sqcap (\neg C) \sqcap (\forall r \cdot D)$ be an axiom. Its associated formula is $\forall x\, ((x = a \vee \exists y_1(p(x, y_1) \wedge A(y_1))) \rightarrow (\exists y_2(q(x, y_2) \wedge B(y_2)) \wedge \neg C(x) \wedge \forall y_3(r(x, y_3) \rightarrow D(y_3))))$. It is equivalent to the conjunction of the three formulas $\mathcal{F}_1 = \forall x\, ((x = a \vee \exists y_1(p(x, y_1) \wedge A(y_1))) \rightarrow \exists y_2(q(x, y_2) \wedge B(y_2)))$, $\mathcal{F}_2 = \forall x\, ((x = a \vee \exists y_1(p(x, y_1) \wedge A(y_1))) \rightarrow \neg C(x))$ and $\mathcal{F}_3 = \forall x\, ((x = a \vee \exists y_1(p(x, y_1) \wedge A(y_1))) \rightarrow \forall y_3(r(x, y_3) \rightarrow D(y_3)))$.

The formula $\mathcal{F}_1$ is translated into the two *dlgp* statements `q(a, Y2), B(Y2).` and `q(X, Y2), B(Y2) :- p(X, Y1), A(Y1).`

The formula $\mathcal{F}_2$ is equivalent to $\forall x\, ((C(x) \wedge (x = a \vee \exists y_1(p(x, y_1) \wedge A(y_1)))) \rightarrow$ `Nothing(x)`). By putting the left side of the implication in disjunctive form, we obtain $\forall x\, (((C(x) \wedge x = a) \vee \exists y_1(p(x, y_1) \wedge A(y_1) \wedge C(x))) \rightarrow$ `Nothing(x)`), that can be translated in the two *dlgp* constraints `! :- C(a).` and `! :- p(X, Y1), A(Y1), C(X).`

Finally, the formula $\mathcal{F}_3$ is equivalent to $\forall y_3\,((\exists x\,(r(x, y_3) \wedge x = a)) \vee (\exists x \exists y_1\,(p(x, y_1) \wedge A(y_1) \wedge r(x, y_3))) \rightarrow D(y_3))$ and can thus be translated into the two *dlgp* rules `D(Y3) :- r(a, Y3).` and `D(Y3):-p(X, Y1), A(Y1), r(X, Y3).`

6 Axioms

We have seen that we can translate into *dlgp* any formula of the form $\forall \vec{x}(\mathcal{B}(\vec{x}) \rightarrow \mathcal{H}(\vec{x}))$, when $\mathcal{B}(\vec{x})$ is a disjunctive formula, and $\mathcal{H}(\vec{x})$ a conjunctive formula.

In Tab. 6, we show that, since the formula associated with a property expression is conjunctive, all OWL 2 axioms that do not require class expressions can be put in such a form. Hence, the following property:

Property 7. OWL 2 axioms with no class expression can be translated into *dlgp*.

On the other hand, an OWL 2 axiom that requires class expressions may not be translatable in *dlgp*. This is why we impose restrictions on all these axioms in the OWL 2 ER profile: `EquivalentClasses` is restricted to EquivClass expressions; `DisjointClasses` and `HasKey` are restricted to SubClass expressions; `ObjectPropertyDomain`, `ObjectPropertyRange`, and `ClassAssertion` are restricted to SuperClass expressions; the first argument of `SubClassOf` must be a SubClass expression and its second argument must be a SuperClass expression. Finally, `DisjointUnion` does not belong to the ER profile.

Assuming these restrictions as displayed in Tab. 7, we conclude with the following property:

Property 8. All OWL 2 axioms in the ER profile can be translated into *dlgp*.

7 OWL 2 profiles

Finally, we point out that the profiles of OWL 2 (namely EL, QL and RL) are fragments of OWL2 ER.

Property 9. All OWL 2 axioms that are either EL, QL or RL axioms are also ER axioms.

We prove that property for each of these profiles.

7.1 OWL 2: the EL profile

Class expressions in OWL 2 EL only use the following constructors: `ObjectSomeValuesFrom`, `ObjectHasValue`, `ObjectHasSelf`, `ObjectOneOf` (restricted to a single individual), `ObjectIntersectionOf`. These constructors form a subset of those listed in Tab. 3, and thus all class expressions in EL are ER EquivClass.

It follows that all axioms (apart from `DisjointUnion`) that can be expressed in EL are ER axioms. Since `DisjointUnion` is excluded from the EL profile, we conclude that any EL axiom is an ER axiom.

Object Property Axioms		
Object Subproperties		
`SubObjectPropertyOf`(p, q)	$p \sqsubseteq q$	$\forall x \forall y\, (\Phi_p(x, y) \rightarrow \Phi_q(x, y))$
Equivalent Object Properties		
`EquivalentObjectProperties`(p, q)	$p \equiv q$	$\forall x \forall y\, (\Phi_p(x, y) \leftrightarrow \Phi_q(x, y))$
Equivalent to the conjunction of $\forall x \forall y\, (\Phi_p(x, y) \rightarrow \Phi_q(x, y))$ *and* $\forall x \forall y\, (\Phi_q(x, y) \rightarrow \Phi_p(x, y))$		
Disjoint Object Properties		
`DisjointObjectProperties`(p, q)	$p \sqsubseteq \neg q$	$\forall x \forall y ((\Phi_p(x, y) \wedge \Phi_q(x, y)) \rightarrow \texttt{Nothing}(x)$
Inverse Object Properties		
`InverseObjectProperties`(p, q)	$p \equiv q^-$	$\forall x \forall y\, (\Phi_p(x, y) \leftrightarrow \Phi_q(y, x))$
Equivalent to the conjunction of $\forall x \forall y\, (\Phi_p(x, y) \rightarrow \Phi_q(y, x))$ *and* $\forall x \forall y\, (\Phi_q(x, y) \rightarrow \Phi_p(y, x))$		
Functional Object Properties		
`FunctionalObjectProperty`(p)		$\forall x \forall y \forall z\, (\Phi_p(x, y) \wedge \Phi_p(x, z) \rightarrow y = z)$
Equivalent to $\forall y \forall z\, (\exists x (\Phi_p(x, y) \wedge \Phi_p(x, z)) \rightarrow y = z)$		
Inverse-Functional Object Properties		
`InverseFunctionalObjectProperty`(p)		$\forall x \forall y \forall z\, (\Phi_p(y, x) \wedge \Phi_p(z, x) \rightarrow y = z)$
Equivalent to $\forall y \forall z\, (\exists x (\Phi_p(y, x) \wedge \Phi_p(z, x)) \rightarrow y = z)$		
Reflexive Object Properties		
`ReflexiveObjectProperty`(p)		$\forall x\, (\texttt{Thing}(x) \rightarrow \Phi_p(x, x))$
Irreflexive Object Properties		
`IrreflexiveObjectProperty`(p)		$\forall x\, (\Phi_p(x, x) \rightarrow \texttt{Nothing}(x))$
Symmetric Object Properties		
`SymmetricObjectProperty`(p)		$\forall x \forall y\, (\Phi_p(x, y) \rightarrow \Phi_p(y, x))$
Asymmetric Object Properties		
`AsymmetricObjectProperty`(p)		$\forall x \forall y ((\Phi_p(x, y) \wedge \Phi_p(y, x) \rightarrow \texttt{Nothing}(x))$
Transitive Object Properties		
`TransitiveObjectProperty`(p)		$\forall x \forall y \forall z\, (\Phi_p(x, y) \wedge \Phi_p(y, z) \rightarrow \Phi_p(x, z))$

Assertions		
Individual Equality		
`SameIndividual`(i_1, i_2)	$i_1 = i_2$	$i_1 = i_2$
Translated by the dlgp *fact* `i1 = i2`		
Individual Inequality		
`DifferentIndividuals`(i_1, i_2)	$i_1 \neq i_2$	$\neg i_1 = i_2$
Translated by the dlgp *constraint* `! :- i1 = i2`		
Positive Object Property Assertions		
`ObjectPropertyAssertion`(i_1, p, i_2)	$p(i_1, i_2)$	$\Phi_p(i_1, i_2)$
Translated by the dlgp *fact obtained by replacing* x *by* i_1 *and* y *by* i_2 *in* $\Phi_p(x, y)$		
Negative Object Property Assertions		
`NegativeObjectPropertyAssertion`(i_1, p, i_2)	$\neg p(i_1, i_2)$	$\neg \Phi_p(i_1, i_2)$
Translated by the dlgp *constraint* `! :-` $\Phi_p(i_1, i_2)$ *as described above.*		

Table 6. OWL 2 axioms that do not require class expressions

7.2 OWL 2: the QL profile

SubClass expressions in OWL 2 QL can only be built from an (atomic) class, or from the constructor `ObjectSomeValuesFrom`, with the added restriction that its second argument is necessarily the class `Thing`. Every QL SubClass expression is thys an ER EquivClass expression (whose associated formula is restricted to a single atom).

SuperClass expressions in QL are built from conjunctions (`ObjectIntersectionOf`) of class expressions that can be either an (atomic) class; the negation (`ObjectComplementOf`) of a SubClassExpression; or obtained from the constructor `ObjectSomeValuesFrom`, with the added restriction that the second argument is an atomic class expression. It follows that every QL Superclass expression is an ER SuperClass expression.

Class Axioms		
Subclass axioms		
SubClassOf(C_1, C_2)	$C_1 \sqsubseteq C_2$	$\forall x\,(\Phi_{C_1}(x) \rightarrow \Phi_{C_2}(x))$
C_1 must be a Subclass expression and C_2 must be a SuperClass expression		
Equivalent Classes		
EquivalentClasses(C_1, C_2)	$C_1 \equiv C_2$	$\forall x\,(\Phi_{C_1}(x) \leftrightarrow \Phi_{C_2}(x))$
Translated by the conjunction of $\forall x\,(\Phi_{C_1}(x) \rightarrow \Phi_{C_2}(x))$ and $\forall x\,(\Phi_{C_2}(x) \rightarrow \Phi_{C_1}(x))$. *Both C_1 and C_2 must be EquivClass expressions.*		
Disjoint Classes		
DisjointClasses(C_1, C_2)	$C_1 \sqsubseteq C_2$	$\forall x\,((\Phi_{C_1}(x) \wedge \Phi_{C_2}(x)) \rightarrow \texttt{Nothing}(x)$
Both C_1 and C_2 must be SubClass expressions.		
Disjoint Union of Class Expressions		
DisjointUnion$(C, C_1, \ldots, C_k)$		$\forall x\,(\Phi_C(x) \leftrightarrow (\vee_{1 \leq i \leq k} \Phi_{C_i}(x)))$ $\wedge_{1 \leq i < j \leq k} \neg(\exists x\,(\Phi_{C_i}(x) \wedge \Phi_{C_j}(x)))$
Cannot be translated into dlgp*, even when restricted to (atomic) classes.*		

Object Property Axioms		
Object Property Domain		
ObjectPropertyDomain(p, C)		$\forall x \forall y\,(\Phi_p(x, y) \rightarrow \Phi_C(x))$
C must be a SuperClass expression.		
Object Property Range		
ObjectPropertyRange(p, C)		$\forall x \forall y\,(\Phi_p(y, x) \rightarrow \Phi_C(x))$
C must be a SuperClass expression.		

Assertions		
Class Assertions		
ClassAssertion(C, i)	$C(i)$	$\Phi_C(i)$
Equivalent to the formula $\forall x\,(x = i \rightarrow \Phi_C(x))$. C must be a SuperClass expression.		

Keys		
HasKey		
HasKey$(C, p_1, \ldots, p_k)$		$\forall x \forall y \forall z_1 \ldots \forall z_k\,((\Phi_C(x) \wedge \Phi_C(y) \wedge_{1 \leq i \leq k} (\Phi_{p_i}(x, z_i) \wedge \Phi_{p_i}(y, z_i))) \rightarrow x = y)$
C must be a SubClass expression.		

Table 7. OWL 2 axioms that require class expressions

Let us now examine the axioms and assertions that can be written in OWL 2 QL. The class axioms `EquivalentClasses` and `DisjointClasses` are restricted to QL SubClass expressions, *i.e.* ER EquivClass expressions. The property axiom `SubObjectPropertyOf` is unrestricted in both ER and QL, while `ObjectPropertyDomain` and `ObjectPropertyRange` have their second argument restricted to a QL SuperClass expression, thus are ER axioms. Assertions allowed in OWL 2 QL are `DifferentIndividuals` and `ObjectPropertyAssertion` (that can always be translated into *dlgp*) and `ClassAssertion`, that is restricted to a QL SubClass expression, *i.e.* an ER EquivClass expression. The axioms `HasKey` and `DisjointUnion` do not appear in OWL 2 QL. The axiom `SubClassOf` is restricted: its first argument must be a QL SubClass expression, while the second must be a QL SuperClass expression. Thus QL `SubClass` axioms are ER `SubClass` axioms.

We conclude that any QL axiom is an ER axiom.

7.3 OWL 2: the RL profile

As ER, OWL 2 RL considers EquivClass, SubClass and SuperClass expressions.

EquivClass expressions are built from the conjunction `ObjectIntersectionOf` of atomic class expressions and the existential restriction `ObjectHasValue`. These constructors form a subset of those listed in Tab. 3, and thus RL EquivClass expressions are ER EquivClass expressions. Since OWL 2 RL restricts the axiom `EquivalentClasses` to EquivClass expressions that can be translated by conjunctive formulas, these axioms are ER axioms.

SubClass expressions are built from the constructors `ObjectIntersectionOf`, `ObjectUnionOf`, `ObjectOneOf`, `ObjectSomeValuesFrom` and `ObjectHasValue`. These constructors form a subset of those listed in Tab. 4, and thus RL SubClass expressions are ER SubClass expressions. Since OWL 2 RL restricts the axioms `DisjointClasses` and `HasKey` to SubClass expressions, these axioms are ER axioms.

SuperClass expressions in RL are defined inductively. A SuperClass expression is either an (atomic) class; the intersection (`ObjectIntersectionOf`) of SuperClass expressions; the complement of (`ObjectComplementOf`) of a SubClass expression; the universal restriction (`ObjectAllValuesFrom`) of a SuperClass expression; or the maximum cardinality (`ObjectMaxCardinality`) of a SubClass expression, when restricted to 0 or 1. It follows that RL SuperClass expressions are ER SuperClass expressions.

Since RL put the same restrictions on axioms as ER, it follows that all RL axioms are ER axioms.

8 Implementation of the translator

When the OWL 2 input belongs to the ER fragment, our tool ensures that it will be translated into a set of existential rules having the same models. We detail here the behavior of our tool when the input does not necessarily belong to the ER fragment.

Each axiom (and assertion) that does not require class expressions (see Tab. 6) is translated into one or two (in the case of `EquivalentObjectProperty` or `InverseObjectProperty`) *dlgp* rules or constraints. Such axioms always belong to the ER fragment.

Each axiom (and assertion) that requires class expressions (except `Disjoint-Union`, that we never handle, for which a warning is issued) is translated into one or two (in the case of `EquivalentClasses`) class inclusions, as described in Tab. 7. For instance, $A \equiv B$ generates the two class inclusions $A \sqsubseteq B$ and $B \sqsubseteq A$; $(\exists R.C)(a)$ generates the class inclusion $\{a\} \sqsubseteq \exists R.C$.

Each class inclusion $A \sqsubseteq B$ thus generated will then be independently analysed. The first step is to rewrite that inclusion in the form $A \sqsubseteq E \sqcap R_1 \sqcap \ldots \sqcap R_k$ where E, if present, is an EquivClass expression and the rests R_i, if present, are neither EquivClass expressions nor an `ObjectIntersectionOf`. The initial class inclusion is thus equivalent to the $k+1$ class inclusions $A \sqsubseteq E$ and, for $1 \leq i \leq k$, $A \sqsubseteq R_i$. We try now to rewrite each inclusion $A \sqsubseteq R_i$. This can be done when R_i is an `ObjectComplementOf`, `ObjectAllValuesFrom`, or `ObjectMax-Cardinality` (0 or 1), and we can replace the inclusion $A \sqsubseteq R_i$ by an inclusion $A' \sqsubseteq R'_i$ as in the proof of Prop. 6. Otherwise that particular class inclusion is not translated and a warning is issued. The whole process is repeated on the inclusion obtained, until the $R_i^{(n)}$ obtained is an EquivClass expression or a warning is issued.

Example 7. Let us consider the class inclusion $A \sqsubseteq (B \sqcup C) \sqcap (\forall r.D)$. Neither $(B \sqcup C)$ nor $(\forall r.D)$ are EquivClass expressions, so we generate the two class inclusions $A \sqsubseteq B \sqcup C$ and $A \sqsubseteq \forall r.D$. We have no possibility to rewrite the first one, so a warning is issued. The second is rewritten into $\exists r^-.A \sqsubseteq D$. Since D is an EquivClass expression, that class inclusion is kept and the analysis halts.

After this first step, the only remaining class inclusions are of form $A \sqsubseteq B$ where B is an EquivClass expression. Their left side are first put into disjunctive normal form to obtain an equivalent inclusion $A_1 \sqcup \ldots \sqcup A_p \sqsubseteq B$ where no A_i is an `ObjectUnionOf`. For each A_i being an EquivClass expression, we generate a *dlgp* expression translating $A_i \sqsubseteq B$, otherwise a warning is issued.

Example 8. Let us consider the class inclusion $A \sqcup \neg B \sqsubseteq \forall r.(C \sqcap \neg B) \sqcap \neg(C \sqcup D) \sqcap \exists r.(B \sqcup C)$. It does not belong to the ER fragment since its left side is not a SubClass expression and its right side is not a SuperClass expression. It is equivalently rewritten into (1) $A \sqcup \neg B \sqsubseteq \forall r.(C \sqcap \neg B)$, (2) $A \sqcup \neg B \sqsubseteq \neg(C \sqcup D)$, and (3) $A \sqcup \neg B \sqsubseteq \exists r.(B \sqcup C)$. (1) is equivalently rewritten into (1.0) $\exists r^-.(A \sqcup \neg B) \sqsubseteq C \sqcap \neg B$ and (2) into (2.0) $(A \sqcup \neg B) \sqcap (C \sqcup D) \sqsubseteq \bot$. Since the right side of (3) is not an EquivClass expression and we don't know how to rewrite it, a warning is issued and that inclusion is not translated. The inclusion (1.0) is equivalently rewritten into (1.0.1) $\exists r^-.(A \sqcup \neg B) \sqsubseteq C$ and (1.0.2) $\exists r^-.(A \sqcup \neg B) \sqsubseteq \neg B$. The inclusion (1.0.2) is equivalently rewritten into (1.0.2.0) $B \sqcap \exists r^-.(A \sqcup \neg B) \sqsubseteq \bot$. Our initial inclusion is thus equivalent to the inclusions (1.0.1), (1.0.2.0), (2.0) and (3). (3) has been rejected and a warning has been issued, and the right side of the other inclusions are EquivClass expressions.

We now put the left sides of (1.0.1), (1.0.2.0), and (2.0) in disjunctive normal form, obtaining the inclusions (1.0.1.0) $\exists r^-.A \sqcup \exists r^-.(\neg B) \sqsubseteq C$, (1.0.2.0.0) $(B \sqcap \exists r^-.A) \sqcup$

$(B \sqcap \exists r^{-}.(\neg B)) \sqsubseteq \bot$ and $(2.0.0)\,(A \sqcap C) \sqcup (A \sqcap D) \sqcup (\neg B \sqcap C) \sqcup (\neg B \sqcap D) \sqsubseteq \bot$. By "splitting" the disjunctions, we obtain the class inclusions $(1.0.1.0.1)\,\exists r^{-}.A \sqsubseteq C$, $(1.0.1.0.2)\,\exists r^{-}.(\neg B) \sqsubseteq C$, $(1.0.2.0.0.1)\,B \sqcap \exists r^{-}.A \sqsubseteq \bot$, $(1.0.2.0.0.2)\,B \sqcap \exists r^{-}.(\neg B) \sqsubseteq \bot$, $(2.0.0.1)\,A \sqcap C \sqsubseteq \bot$, $(2.0.0.2)\,A \sqcap D \sqsubseteq \bot$, $(2.0.0.3)\,\neg B \sqcap C \sqsubseteq \bot$ and $(2.0.0.4)\,\neg B \sqcap D \sqsubseteq \bot$. The left side of axioms (1.0.1.0.2), (1.0.2.0.0.2), (2.0.0.3) and $(2.0.0.4)$ are not EquivClass expressions, so they cannot be translated and four warnings are issued. The other axioms are translated into *dlgp*.

$(1.0.1.0.1)$ `C(X) :- r(Y, X), A(Y)`
$(1.0.2.0.0.1)$ `! :- B(X), r(Y, X), A(X)`
$(2.0.0.1)$ `! :- A(X), C(X)`
$(2.0.0.2)$ `! :- A(X), D(X)`

The initial class inclusion (that does not belong to ER) has been translated into nine class inclusions, from which four could be translated into *dlgp*. Five warnings have been issued.

When the input belongs to the OWL 2 ER fragment, no warning can be issued and the models of the OWL 2 ontology and the models of its *dlgp* translation are the same. However, even when a warning is issued, our algorithm ensures that all models of the OWL 2 ontology are models of the *dlgp* translation.

9 conclusion

In this report, we presented the OWL 2 ER profile, which allows to translate the "Datalog+" part of an OWL 2 ontology into *dlgp*. The associated software and documentation can be found at `https://graphik-team.github.io/graal/owl2dlgp`. Future improvements will be made available on the same website.

Appendix: Grammar for the ER profile

```
Class := IRI

Datatype := IRI

ObjectProperty := IRI

DataProperty := IRI

AnnotationProperty := IRI

Individual := NamedIndividual | AnonymousIndividual

NamedIndividual := IRI

AnonymousIndividual := nodeID

Literal := typedLiteral | stringLiteralNoLanguage | stringLiteralWithLanguage
typedLiteral := lexicalForm '^^' Datatype
lexicalForm := quotedString
stringLiteralNoLanguage := quotedString
stringLiteralWithLanguage := quotedString languageTag
```

```
ObjectPropertyExpression := ObjectProperty | InverseObjectProperty

InverseObjectProperty := 'ObjectInverseOf' '(' ObjectProperty ')'

DataPropertyExpression := DataProperty



ZeroOrOne  := '0' | '1'

AtomicClassExpression :=
    Class |
    SimpleObjectSomeValuesFrom | ObjectHasValue | ObjectHasSelf |
    SimpleObjectOneof | SimpleObjectMinCardinality |
    DataHasValue | SimpleDataMinCardinality

SimpleObjectSomeValuesFrom :=
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression owl:Thing ')'

ObjectHasValue := 'ObjectHasValue' '(' ObjectPropertyExpression Individual ')'

ObjectHasSelf := 'ObjectHasSelf' '(' ObjectPropertyExpression ')'

SimpleObjectOneOf := 'ObjectOneOf' '(' Individual ')'

SimpleObjectMinCardinality :=
    'ObjectMinCardinality' '(' ZeroOrOne ObjectPropertyExpression ')'

DataHasValue := 'DataHasValue' '(' DataPropertyExpression Literal ')'

SimpleDataMinCardinality :=
    'DataMinCardinality' '(' ZeroOrOne DataPropertyExpression ')'



EquivClassExpression :=
    AtomicClassExpression |
    EquivObjectIntersectionOf |
    EquivObjectSomeValuesFrom |
    EquivObjectMinCardinality |
    EquivDataSomeValuesFrom |
    EquivDataMinCardinality

EquivObjectIntersectionOf :=
    'ObjectIntersectionOf' '(' EquivClassExpression EquivClassExpression
                               { EquivClassExpression } ')'

EquivObjectSomeValuesFrom :=
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression EquivClassExpression ')'

EquivObjectMinCardinality :=
    'ObjectMinCardinality' '(' ZeroOrOne ObjectPropertyExpression EquivClassExpression ')'

EquivDataSomeValuesFrom :=
    'DataSomeValuesFrom' '(' DataPropertyExpression { DataPropertyExpression }
                             EquivDataRange ')'

EquivDataMinCardinality :=
    'DataMinCardinality' '(' ZeroOrOne DataPropertyExpression EquivDataRange ')'

EquivDataRange :=
    Datatype |
    EquivDataIntersectionOf |
    EquivDataOneOf

EquivDataIntersectionOf := 'DataIntersectionOf' '(' EquivDataRange EquivDataRange
                                               { EquivDataRange } ')'
```

```
EquivDataOneOf := 'DataOneOf' '(' Literal ')'



SubClassExpression :=
    AtomicClassExpression |
    SubObjectIntersectionOf |
    SubObjectSomeValuesFrom |
    SubObjectMinCardinality |
    SubObjectUnionOf |
    SubObjectOneOf
    SubDataSomeValuesFrom |
    SubDataMinCardinality

SubObjectIntersectionOf :=
    'ObjectIntersectionOf' '(' SubClassExpression SubClassExpression
                               { SubClassExpression } ')'

SubObjectSomeValuesFrom :=
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression SubClassExpression ')'

SubObjectMinCardinality :=
    'ObjectMinCardinality' '(' ZeroOrOne ObjectPropertyExpression SubClassExpression ')'

SubObjectUnionOf :=
    'ObjectUnionOf' '(' SubClassExpression SubClassExpression { SubClassExpression } ')'

SubObjectOneOf := 'ObjectOneOf' '(' Individual { Individual }')'

SubDataSomeValuesFrom :=
    'DataSomeValuesFrom' '(' DataPropertyExpression { DataPropertyExpression }
                             SubDataRange ')'

SubDataMinCardinality :=
    'DataMinCardinality' '(' ZeroOrOne DataPropertyExpression SubDataRange ')'

SubDataRange :=
    Datatype |
    SubDataIntersectionOf |
    SubDataUnionOf |
    SubDataOneOf

SubDataIntersectionOf := 'DataIntersectionOf' '(' SubDataRange SubDataRange { SubDataRange } ')'

SubDataUnionOf := 'DataUnionOf' '(' SubDataRange SubDataRange { SubDataRange } ')'

SubDataOneOf := 'DataOneOf' '(' Literal { Literal } ')'



SuperClassExpression :=
    AtomicClassExpression |
    SuperObjectIntersectionOf |
    SuperObjectSomeValuesFrom |
    SuperObjectAllValuesFrom |
    SuperObjectComplementOf |
    SuperObjectMinCardinality |
    SuperObjectMaxCardinality |
    SuperObjectExactCardinality |
    SuperDataSomeValuesFrom |
    SuperDataAllValuesFrom |
    SuperDataMinCardinality |
    SuperDataMaxCardinality |
    SuperDataExactCardinality

SuperObjectIntersectionOf :=
    'ObjectIntersectionOf' '(' SuperClassExpression SuperClassExpression
```

```
                              { SuperClassExpression } ')'

SuperObjectSomeValuesFrom :=
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression EquivClassExpression ')'

SuperObjectAllValuesFrom :=
    'ObjectAllValuesFrom' '(' ObjectPropertyExpression SuperClassExpression ')'

SuperObjectComplementOf := 'ObjectComplementOf' '(' SubClassExpression ')'

SuperObjectMinCardinality :=
    'ObjectMinCardinality' '(' ZeroOrOne ObjectPropertyExpression
                              EquivClassExpression ')'

SuperObjectMaxCardinality :=
    'ObjectMaxCardinality' '(' ZeroOrOne ObjectPropertyExpression
                              [ SubClassExpression ] ')'

SuperObjectExactCardinality :=
    'ObjectExactCardinality' '(' ZeroOrOne ObjectPropertyExpression
                                [ EquivClassExpression ] ')'

SuperDataSomeValuesFrom :=
    'DataSomeValuesFrom' '(' DataPropertyExpression { DataPropertyExpression }
                            EquivDataRange ')'

SuperDataAllValuesFrom :=
    'DataAllValuesFrom' '(' DataPropertyExpression { DataPropertyExpression }
                            SuperDataRange ')'

SuperDataMinCardinality :=
    'DataMinCardinality' '(' ZeroOrOne DataPropertyExpression EquivDataRange ')'

SuperDataMaxCardinality :=
    'DataMaxCardinality' '(' ZeroOrOne DataPropertyExpression [ SubDataRange ] ')'

SuperDataExactCardinality :=
    'DataExactCardinality' '(' ZeroOrOne DataPropertyExpression [ EquivDataRange ] ')'

SuperDataRange :=
    Datatype |
    SuperDataIntersectionOf |
    SuperDataComplementOf |

SuperDataIntersectionOf :=
    'DataIntersectionOf' '(' SuperDataRange SuperDataRange { SuperDataRange } ')'

SuperDataComplementOf := 'DataComplementOf' '(' SubDataRange ')'



Axiom :=
    Declaration |
    ClassAxiom |
    ObjectPropertyAxiom |
    DataPropertyAxiom |
    DatatypeDefinition |
    HasKey |
    Assertion |
    AnnotationAxiom



ClassAxiom := SubClassOf | EquivalentClasses | DisjointClasses

SubClassOf :=
    'SubClassOf' '(' axiomAnnotations SubClassExpression SuperClassExpression ')'
```

```
EquivalentClasses :=
    'EquivalentClasses' '(' axiomAnnotations EquivClassExpression
                            EquivClassExpression { EquivClassExpression } ')'

DisjointClasses :=
    'DisjointClasses' '(' axiomAnnotations SubClassExpression SubClassExpression
                          { SubClassExpression } ')'



ObjectPropertyAxiom :=
    SubObjectPropertyOf | EquivalentObjectProperties |
    DisjointObjectProperties | InverseObjectProperties |
    ObjectPropertyDomain | ObjectPropertyRange |
    FunctionalObjectProperty | InverseFunctionalObjectProperty |
    ReflexiveObjectProperty | IrreflexiveObjectProperty |
    SymmetricObjectProperty | AsymmetricObjectProperty |
    TransitiveObjectProperty

SubObjectPropertyOf :=
    'SubObjectPropertyOf' '(' AxiomAnnotations subObjectPropertyExpression
                              superObjectPropertyExpression ')'

subObjectPropertyExpression := ObjectPropertyExpression | propertyExpressionChain

propertyExpressionChain :=
    'ObjectPropertyChain' '(' ObjectPropertyExpression ObjectPropertyExpression
                              { ObjectPropertyExpression } ')'

superObjectPropertyExpression := ObjectPropertyExpression

EquivalentObjectProperties :=
    'EquivalentObjectProperties' '(' axiomAnnotations ObjectPropertyExpression
                                     ObjectPropertyExpression { ObjectPropertyExpression } ')'

DisjointObjectProperties :=
    'DisjointObjectProperties' '(' axiomAnnotations ObjectPropertyExpression
                                   ObjectPropertyExpression { ObjectPropertyExpression } ')'

ObjectPropertyDomain :=
    'ObjectPropertyDomain' '(' axiomAnnotations ObjectPropertyExpression
                               SuperClassExpression ')'

ObjectPropertyRange :=
    'ObjectPropertyRange' '(' axiomAnnotations ObjectPropertyExpression
                              SuperClassExpression ')'

InverseObjectProperties :=
    'InverseObjectProperties' '(' axiomAnnotations ObjectPropertyExpression
                                  ObjectPropertyExpression ')'

FunctionalObjectProperty :=
    'FunctionalObjectProperty' '(' axiomAnnotations ObjectPropertyExpression ')'

InverseFunctionalObjectProperty :=
    'InverseFunctionalObjectProperty' '(' axiomAnnotations ObjectPropertyExpression ')'

ReflexiveObjectProperty :=
    'ReflexiveObjectProperty' '(' axiomAnnotations ObjectPropertyExpression ')'

IrreflexiveObjectProperty :=
    'IrreflexiveObjectProperty' '(' axiomAnnotations ObjectPropertyExpression ')'

SymmetricObjectProperty :=
    'SymmetricObjectProperty' '(' axiomAnnotations ObjectPropertyExpression ')'

AsymmetricObjectProperty :=
    'AsymmetricObjectProperty' '(' axiomAnnotations ObjectPropertyExpression ')'
```

```
TransitiveObjectProperty :=
    'TransitiveObjectProperty' '(' axiomAnnotations ObjectPropertyExpression ')'



DataPropertyAxiom :=
    SubDataPropertyOf | EquivalentDataProperties | DisjointDataProperties |
    DataPropertyDomain | DataPropertyRange | FunctionalDataProperty

SubDataPropertyOf :=
    'SubDataPropertyOf' '(' axiomAnnotations subDataPropertyExpression
                                             superDataPropertyExpression ')'
subDataPropertyExpression := DataPropertyExpression
superDataPropertyExpression := DataPropertyExpression

EquivalentDataProperties :=
    'EquivalentDataProperties' '(' axiomAnnotations DataPropertyExpression DataPropertyExpression
                                                    { DataPropertyExpression } ')'

DisjointDataProperties :=
    'DisjointDataProperties' '(' axiomAnnotations DataPropertyExpression DataPropertyExpression
                                                  { DataPropertyExpression } ')'

DataPropertyDomain :=
    'DataPropertyDomain' '(' axiomAnnotations DataPropertyExpression SuperClassExpression ')'

DataPropertyRange :=
    'DataPropertyRange' '(' axiomAnnotations DataPropertyExpression SuperDataRange ')'

FunctionalDataProperty := 'FunctionalDataProperty' '(' axiomAnnotations DataPropertyExpression ')'



DatatypeDefinition := 'DatatypeDefinition' '(' axiomAnnotations Datatype EquivDataRange ')'



HasKey :=
    'HasKey' '(' axiomAnnotations SubClassExpression '(' { ObjectPropertyExpression } ')'
                 '(' { DataPropertyExpression } ')' ')'



Assertion :=
    SameIndividual | DifferentIndividuals | ClassAssertion |
    ObjectPropertyAssertion | NegativeObjectPropertyAssertion |
    DataPropertyAssertion | NegativeDataPropertyAssertion

sourceIndividual := Individual
targetIndividual := Individual
targetValue := Literal

SameIndividual :=
    'SameIndividual' '(' axiomAnnotations Individual Individual { Individual } ')'

DifferentIndividuals :=
    'DifferentIndividuals' '(' axiomAnnotations Individual Individual { Individual } ')'

ClassAssertion :=
    'ClassAssertion' '(' axiomAnnotations SuperClassExpression Individual ')'

ObjectPropertyAssertion :=
    'ObjectPropertyAssertion' '(' axiomAnnotations ObjectPropertyExpression
                                  sourceIndividual targetIndividual ')'

NegativeObjectPropertyAssertion :=
    'NegativeObjectPropertyAssertion' '(' axiomAnnotations ObjectPropertyExpression
```

```
                                         sourceIndividual targetIndividual ')'

DataPropertyAssertion :=
    'DataPropertyAssertion' '(' axiomAnnotations DataPropertyExpression
                                sourceIndividual targetValue ')'

NegativeDataPropertyAssertion :=
    'NegativeDataPropertyAssertion' '(' axiomAnnotations DataPropertyExpression
                                        sourceIndividual targetValue ')'
```