# Datalog+, RuleML and OWL 2: Formats and Translations for Existential Rules

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

# Datalog+, RuleML and OWL 2: Formats and Translations for Existential Rules

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

Inria, CNRS and University of Montpellier
France

**Abstract.** This paper is devoted to formats and translations for Datalog+. We first introduce the *dlgp* format, which extends classical Datalog format to Datalog+. It allows to encode facts, existential rules (including equality), negative constraints and conjunctive queries. Moreover, for compatibility with Semantic Web languages, this format includes Web notions (IRIs and literals, according to Turtle syntax). Second, we define a translation from *dlgp* to the Datalog+ fragment of RuleML. Third, we define a translation from OWL 2 to *dlgp*. We point out that the composition of both translations allows to import OWL 2 to RuleML. The associated parsers and translators are available.

## 1 Introduction

The novel framework of existential rules, also known as Datalog+ (and Datalog$\pm$ for its decidable fragments) raised considerable interest in the last years, specially in the context of ontology-mediated query answering. For the theoretical foundations of this framework, we refer the reader to [CGK08,BLMS09,CGL09,BLMS11,KR11,CGP12]. A sublanguage of RuleML corresponding to Datalog+ has been recently defined (see Deliberation RuleML 1.01).[1]

This paper is devoted to formats and translators for the existential rule framework. It can be seen as a companion to the paper [BLM$^+$15] devoted to Graal, a toolkit for querying existential rule knowledge bases. First, we introduce a textual format, called *dlgp*, for "Datalog+". This format is indeed an extension of the classical Datalog format to Datalog+. In addition, to ensure compatibility with Semantic Web languages, the current version (2.0) includes Web notions like IRIs and literals. Second, we briefly present a translation from *dlgp* to (the Datalog+ sublanguage of) RuleML, which is quite direct. Third, we define a translation from OWL 2 to *dlgp*. More precisely, we define OWL 2 ER, an OWL 2 profile that can be translated into the existential rule framework. In particular, this profile generalizes so-called tractable profiles of OWL 2. Our translator from OWL 2 to *dlgp* accepts as input any OWL 2 file but ignores (parts of) OWL 2 axioms that cannot be recast as axioms of OWL 2 ER. Note that the composition of both translations allows to import OWL 2 to RuleML.

The *dlgp* parser, as well as translators from *dlgp* to RuleML and from OWL 2 to *dlgp* are available online at `http://graphik-team.github.io/graal/` (Graal homepage). The paper focuses on salient aspects of these three points and illustrates them

[1] `http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.01`

by examples. Details on the *dlgp* format and OWL 2 ER profile (including OWL 2 ER grammar) can be found in technical reports available on Graal website.

**Preliminaries.** We recall here the basic components of the existential rule framework: *facts*, *(pure) existential rules*, which include equality rules, *negative constraints* and *conjunctive queries*. We consider logical atoms of the form $p(t_1 \dots t_n)$, where $p$ is a *predicate* of any arity greater or equal to one (i.e., $n \geq 1$) and the $t_i$ are variables or constants. We also consider equality atoms (of the form $t_1 = t_2$). By *conjunction*, we mean a conjunction of such atoms.

- A *(pure) existential rule* is a positive rule of the form $\forall X \forall Y (\mathcal{B}[X, Y] \rightarrow \exists Z \, \mathcal{H}[X, Z])$, where $X$, $Y$ and $Z$ are sets of variables, $\mathcal{B}$ (the body) is a conjunction with variables in $X$ and $Y$, and $\mathcal{H}$ (the head) is a conjunction with variables in $X$ and $Z$; hence $Z$ denotes the set of variables that occur in $\mathcal{H}$ but not in $\mathcal{B}$; a specific kind of pure existential rules are *equality rules*, whose head is restricted to an equality atom.
- A *fact* is an existentially closed conjunction, i.e., of the form $\exists X \mathcal{F}[X]$, where $\mathcal{F}$ is a conjunction whose set of variables is exactly $X$. Note that we generalize here the classical notion of a fact defined as a ground atom. Indeed, a fact is usually seen as a rule with an empty body, hence the previous definition is in line with existential rules. Moreover, existential variables in facts allow to naturally encode unknown values in databases ("nulls") or blank nodes in RDF.
- A *negative constraint* is the negation of a fact, i.e., of the form $\neg \exists X \mathcal{C}[X]$; equivalently, it can be seen as a rule with a head restricted to the always-false symbol $\bot$: $\forall X (\mathcal{C}[X] \rightarrow \bot)$;
- A *conjunctive query* is an existentially quantified conjunction, in which the free variables are called answer variables. It can also be seen as a rule of the form $\forall X \forall Y (\mathcal{B}[X, Y] \rightarrow ans(X))$, where $ans$ is a special predicate and $X$ is the set of answer variables.

The following running example illustrates these different components.

*Example 1 (Running example).* We consider a vocabulary composed of the following predicates, with their arity mentioned after their name: Project/1, Researcher/1, isProject/3, hasCost/2, hasExpertise/2, isMember/2. The unary predicates can be seen as types of entities (i.e., classes or concepts); the ternary predicate *isProject* relates a project, the area of this project and the leader of this project; the predicate *hasCost* relates a project and its cost, while predicates *hasExpertise* and *isMember* relate a researcher to an area and to a project, respectively.

We consider a knowledge base built on this vocabulary composed of the following three facts and four rules:

- $F_1 = Researcher(a)$, where $a$ is a constant
  *"Individual $a$ is a researcher"*
- $F_2 = \exists X \exists Y (isMember(a, X) \wedge isProject(X, kr, Y) \wedge hasCost(X, 1.5))$,
  where $a$, $kr$ and 1.5 are constants;
  *"Individual $a$ is member of a project in the* kr *area which has a cost of 1.5"*

- $F_3 = Researcher(b) \wedge hasExpertise(b, sw)$, where $b$ and $sw$ are constants
  *"Individual b is a researcher who has expertise in the* sw *area"*
- $R_1 = \forall X \forall Y \forall Z(isProject(X, Y, Z) \rightarrow isMember(Z, X))$
  *"Every leader of a project is a member of this project"*
- $R_2 = \forall X \forall Y (Researcher(X) \wedge hasExpertise(X, Y) \rightarrow \exists Z \exists L\, (isProject(Z, Y, L) \wedge isMember(X, Z)))$
  *"Every researcher expert in an area is a member of a project in that area"*
- $R_3 = \forall X \forall Y \forall Z(isProject(X, Y, Z) \wedge isProject(X, Y, Z') \rightarrow Z = Z')$
  *"Every project has at most one leader"*
- $R_4 = \forall X(Researcher(X) \wedge Project(X) \rightarrow \bot)$
  *"Classes Researcher and Project are disjoint"*

$R_1$ is a (plain) Datalog rule since it has no existential variable. $R_3$ is an equality rule. $R_4$ is a negative constraint. Note that Fact $F_3$ can be decomposed into two atomic facts, whereas $F_2$ cannot because its atoms are linked by the variable $X$. Here are two examples of conjunctive queries on this vocabulary:

- $Q_1 = \exists Y \exists Z(Researcher(X) \wedge isMember(X, Y) \wedge isProject(Y, kr, Z))$
  *"find all researchers members of a project in* kr*"*
- $Q_2 = \exists X \exists Z isProject(X, sw, Z)$
  *"is there a project in the* sw *area ?"*

The set of answers to $Q_1$ on the knowledge base is $\{a\}$ (found in $F_1 \wedge F_2$). The answer to $Q_2$ is *yes* (which requires to consider $F_3 \wedge R_2$).

In the next sections, we present the *dlgp* format, then its translation to RuleML, and finally the translation from OWL 2 to *dlgp*.

## 2 The Datalog+ format (dlgp)

The *dlgp* format is a textual exchange format, meant to be at once human-friendly, concise and easy to parse. It can be seen as an extension of the commonly used format for plain Datalog. In particular, rules are of the form `head :- body`. Note that a comma is interpreted as a logical $\wedge$, even in a rule head. Quantifiers are omitted since there is no ambiguity. We remind that variables that occur in the head and not in the body are existentially quantified.

In Datalog, variables begin with an uppercase letter, while constants and predicates begin with a lowercase letter. To make the *dlgp* format compatible with data from the Semantic Web, more elaborate kinds of identifiers are mandatory. That is why *dlgp* deals with the notions of IRI and literal, as detailed later.

The following example shows a *dlgp* encoding of the knowledge from Example 1. The predicates that begin with an uppercase letter are enclosed in angle brackets (<>). The directives `@facts`, `@rules`, `@constraints`, `@queries` are optional indications for the reader or the parser. [2] Knowledge elements are preceded by an optional label (enclosed in square brackets).

[2] These directives allow the parser to be sooner informed of the nature of coming statements. However, the parser is free to ignore them.

*Example 2 (Running example in* dlgp*).*

```
% this dlgp file encodes Example 1
@facts
[F1] <Researcher>(a).
[F2] isMember(a,X), isProject(X, kr, Y), hasCost(X,1.5).
[F3] <Researcher>(b), hasExpertise(b,sm).
@rules
[R1] isMember(Z,X) :- isProject(X,Y,Z).
[R2] isProject(Z,Y,L), isMember(X,Z) :- <Researcher>(X), hasExpertise(X,Y).
[R3] Z1=Z2 :- isProject(X,Y,Z1), isProject(X,Y,Z2).
@constraints
[R_4] ! :- <Researcher>(X), <Project>(X).
@queries
[Q1] ? (X) :- isMember(X,Y), isProject(Y, kr, Z).
[Q2] ? :- isProject(X,sw,Z).
```

Other features of *dlgp* have been added for compatiblity with Semantic Web languages:

- predicates and constants can be denoted by IRIs;
- constants can be literals;
- namespaces are introduced;
- a universal class can be declared (e.g., Thing in OWL 2);
- whether the Unique Name Assumption is made or not can be specified.

To implement these features, we rely on Turtle syntax.[3] In particular, a predicate or a constant can be denoted by an IRI in Turtle format, which can be written in three forms: a relative IRI, an absolute IRI or a prefixed name. Relative and absolute IRIs are strings of the form `<iri>` (`IRIREF` in Turtle grammar). A prefixed name is a string prefixed by a namespace (`PrefixedName` in Turtle grammar); moreover, the classical Datalog notation (a string beginning with a lowercase letter) is allowed and understood as a relative IRI. A constant can also be a literal (`literal` in Turtle grammar).

For instance, here are different ways of writing a predicate:

- `pred` (classical Datalog notation);
- `<pred>` (a relative IRI, whose base is specified by a directive of the form `@base <http://example.org>`);
- `<http://exemple.org/pred>` (an absolute IRI)
- `ex:pred` (a prefixed name; the IRI associated with the prefix `ex` is specified by a directive of the form `@prefix ex: <http://exemple.org/>`

All these forms can be used to encode a constant. Moreover, constants can be described by *literals*, e.g., `-5.1`, `true`, `"constant"`, or any other Turtle literal. Note that the tokens `true` and `false` are interpreted as Boolean literals and not as IRIs.

The following example shows another *dlgp* encoding of the knowledge from Example 1 using IRIs and namespaces.

*Example 3 (Running example in* dlgp *with IRIs and namespaces).*

[3] `http://www.w3.org/TR/turtle/`

```
@prefix ex: <http://example.org/> % namespace
@facts
[F1] ex:Researcher(ex:a).
[F2] ex:isMember(ex:a,X), ex:isProject(X, ex:kr, Y), ex:hasCost(X,1.5).
[F3] ex:Researcher(ex:b), ex:hasExpertise(ex:b,ex:sw).
@rules
[R1] ex:isMember(Z,X) :- ex:isProject(X,Y,Z).
[R2] ex:isProject(Z,Y,L), ex:isMember(X,Z) :- ex:Researcher(X),
ex:hasExpertise(X,Y).
[R3] Z1=Z2 :- ex:isProject(X,Y,Z1), ex:isProject(X,Y,Z2).
@constraints
[R_4] ! :- ex:Researcher(X), ex:Project(X).
@queries
[Q1] ? (X) :- ex:isMember(X,Y), ex:isProject(Y, kr, Z).
[Q2] ? :- ex:isProject(X,ex:sw,Z).
```

The complete grammar of *dlgp* is available on Graal website.

## 3 From dlgp to RuleML

The translation from *dlgp* to RuleML targets more precisely the Datalog+ fragment of Deliberation RuleML 1.01. This sublanguage includes:

- positive facts,
- universally quantified implications,
- existentials in the heads of implications,
- equality,
- falsity in the heads of implications,
- conjunctions in the heads of implications.

The translation from *dlgp* to this sublanguage is quite direct. Since existential quantification in facts is not allowed in this sublanguage, we translate each existential variable in facts by a new constant [4] , as illustrated in Example 4. This example also illustrates the translation of datatypes.

*Example 4 (Fact with existential variables and datatypes).*

```
[F2] isMember(a,X), isProject(X, kr, Y), hasCost(X, 1.5).
```

```
<Assert>
  <Atom><Rel>isMember</Rel><Ind>a</Ind><Ind>i0</Ind></Atom>
  <Atom><Rel>isProject</Rel><Ind>i0</Ind><Ind>kr</Ind><Ind>i1</Ind></Atom>
  <Atom><Rel>hasCost</Rel><Ind>i0</Ind>
    <Data xsi:type="xs:decimal">1.5</Data></Atom>
</Assert>
```

Example 5 illustrates the translation of existential rules. Constraints are translated in the same manner with an empty disjunction in the head.

*Example 5 (Rule with existential variables in the head).*

[4] To preserve the semantics of the knowledge base, the unique name assumption should not be made on these new constants.

```
[R2] isProject(Z,Y,L), isMember(X,Z) :- <Researcher>(X), hasExpertise(X,Y).

<Assert><!-- R2 -->
  <Forall><Var>X</Var><Var>Y</Var>
    <Implies><if><And>
      <Atom><Rel>Researcher</Rel><Var>X</Var></Atom>
      <Atom><Rel>hasExpertise</Rel><Var>X</Var><Var>Y</Var></Atom>
    </And></if><then>
      <Exists><Var>L</Var><Var>Z</Var><And>
        <Atom><Rel>isProject</Rel><Var>Z</Var><Var>Y</Var><Var>L</Var></Atom>
        <Atom><Rel>isMember</Rel><Var>X</Var><Var>Z</Var></Atom>
      </And></Exists>
    </then></Implies>
  </Forall>
</Assert>
```

Example 6 illustrates the translation of queries.

*Example 6 (Query).*

```
[Q1] ? (X) :- isMember(X,Y), isProject(Y, kr, Z).

<Query><!-- Q1 -->
  <Exists><Var>Y</Var><Var>Z</Var><And>
    <Atom><Rel>isMember</Rel><Var>X</Var><Var>Y</Var></Atom>
    <Atom><Rel>isProject</Rel><Var>Y</Var><Ind>kr</Ind><Var>Z</Var></Atom>
  </And></Exists>
</Query>
```

Finally, Example 7 shows how IRIs are translated.

*Example 7 (Fact with IRI).*

```
@prefix ex: <http://example.com/>
[F1] ex:Researcher(ex:a).

<Assert>
  <Atom>
    <Rel iri="http://example.com/Researcher"/>
    <Ind iri="http://example.com/a"/>
  </Atom>
</Assert>
```

The full translation of Example 3 to RuleML is available on Graal website.

## 4 From OWL 2 to dlgp: the ER profile

We introduce here the ER (for Existential Rule) profile of OWL 2, for which all axioms can be translated into *dlgp* 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 space requirements and 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.

### 4.1 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 $p$.[5]

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 $y$".

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 8.

*Example 8.* 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. 4.2. *SubClass* expressions can only appear in the left side (Sect. 4.3), while *SuperClass* expressions can only appear in the right side (Sect. 4.4). We show in Sect. 4.5 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.

[5] 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`.

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

### 4.2 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 $\vec{x} \cup \vec{z}$.[6]

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

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

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

*Example 9.* The class expression $\exists p \cdot (\exists q \cdot C)$ is translated into FOL by $\Phi_{\exists p \cdot (\exists q \cdot C)}(x) = \exists y_1(p(x,y_1) \wedge (\exists y_2(q(y_1,y_2) \wedge C(y_2))))$. By putting it in prenex form, we obtain the conjunctive formula $\exists y_1 \exists y_2(p(x,y_1) \wedge q(y_1,y_2) \wedge C(y_2))$. 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 10.* The axiom $A \sqsubseteq \exists p \cdot \bot$ is translated by the FOL formula $\forall x(A(x) \rightarrow (\exists y(p(x,y) \wedge \texttt{Nothing}(y))))$, 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 $\forall x\,((x = a) \rightarrow \exists y(p(x,y) \wedge C(y))$ and thus can be expressed by the *dlgp* fact: `p(a, Y), C(Y).`

[6] 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.

| **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$(p)$ | $\exists p \cdot$ 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

### 4.3 SubClass expressions

A FOL formula $\mathcal{F}(\vec{x})$ is said to be *disjunctive* when it is a disjunction $\mathcal{F}_1(\vec{x}) \vee \ldots \vee \mathcal{F}_k(\vec{x})$ of conjunctive formulas. In that case, we say that the disjunction is of size $k$.[7]

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

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

*Example 11.* 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.

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

| **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`$(C_1, \ldots, C_k)$ | $C_1 \sqcup \ldots \sqcup C_k$ | $\Phi_{C_1}(x) \vee \ldots \vee \Phi_{C_k}(x)$ |
| **Enumeration of individuals (unrestricted)** | | |
| `ObjectOneOf`$(i_1, \ldots, i_k)$ | $\{i_1, \ldots, i_k\}$ | $x = i_1 \vee \ldots \vee x = i_k$ |

**Table 4.** SubClass expressions constructors.

*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.

*Example 12.* 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).`

### 4.4 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.

*Proof.* We show the 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.

| **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.

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 $\forall x\,(\exists y\,(\Phi_B(x) \wedge \Phi_p(x,y) \wedge \Phi_{H'}(y)) \rightarrow \texttt{Nothing}(x))$. 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 13.* 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).`

### 4.5 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.

| **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$(p, q)$ | $p \sqsubseteq \neg q$ | $\forall x \forall y((\Phi_p(x, y) \wedge \Phi_q(x, y)) \rightarrow \mathtt{Nothing}(x)$ |
| **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$(p)$ | | $\forall x\, (\mathtt{Thing}(x) \rightarrow \Phi_p(x, x))$ |
| **Irreflexive Object Properties** | | |
| IrreflexiveObjectProperty$(p)$ | | $\forall x\, (\Phi_p(x, x) \rightarrow \mathtt{Nothing}(x))$ |
| **Symmetric Object Properties** | | |
| SymmetricObjectProperty$(p)$ | | $\forall x \forall y\, (\Phi_p(x, y) \rightarrow \Phi_p(y, x))$ |
| **Asymmetric Object Properties** | | |
| AsymmetricObjectProperty$(p)$ | | $\forall x \forall y((\Phi_p(x, y) \wedge \Phi_p(y, x) \rightarrow \mathtt{Nothing}(x))$ |
| **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

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*.

| **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 $\forall x\,(\Phi_{C_2}(x) \rightarrow \Phi_{C_1}(x))$.<br>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$(C, C_1, \ldots, C_k)$ | | $\forall x\,(\Phi_C(x) \leftrightarrow (\vee_{1\leq i\leq k}\Phi_{C_i}(x)))$<br>$\wedge_{1\leq i<j\leq k}\neg(\exists x\,(\Phi_{C_i}(x) \wedge \Phi_{C_j}(x)))$ |
| *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

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.

### 4.6 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 `DisjointUnion`, 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 `ObjectMaxCardinality` (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 14.* 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 15.* 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.

## 5 Conclusion

In this paper, we presented the main aspects of the *dlgp* format dedicated to the framework of existential rules, the translation from *dlgp* to the Datalog+ fragment of RuleML and the OWL 2 ER profile, which allows to translate the "Datalog+" part of an OWL 2 ontology (as precisely explained in the preceding section). The associated software tools and documentation can be found on Graal website. Future improvements will be made available on the same website.

## References


BLM+15. J.-F. Baget, M. Leclère, M.-L. Mugnier, S. Rocher, and C. Sipieter. Graal: A Toolkit for Query Answering with Existential Rules. In *RuleML, to appear*, 2015.

BLMS09. J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending decidable cases for rules with existential variables. In *IJCAI'09*, pages 677–682, 2009.

BLMS11. J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9-10):1620–1654, 2011.

CGK08. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR'08*, pages 70–80, 2008.

CGL09. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS'09*, pages 77–86, 2009.

CGP12. A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

KR11. M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI'11*, pages 963–968, 2011.