Model with DLs + Solve with ASP!

A case study from Concept Learning

Francesca A. Lisi

Dipartimento di Informatica &
Centro Interdipartimentale di Logica e Applicazioni (CILA)
Università degli Studi di Bari "Aldo Moro", Italy
francesca.lisi@uniba.it

Abstract. Research in Machine Learning (ML) has traditionally focussed on designing effective algorithms for solving particular tasks. However, there is an increasing interest in providing the user with a means for specifying what the ML problem in hand actually is rather than letting him struggle to outline how the solution to that problem needs to be computed. This corresponds to a *model+solver* approach to ML, in which the user specifies the problem in a *declarative modeling language* and the system automatically transforms such models into a format that can be used by a *solver* to efficiently generate a solution. In this paper, we propose a *model+solver* approach to Concept Learning problems which combines the efficacy of *Description Logics* (DLs) in conceptual modeling with the efficiency of *Answer Set Programming* (ASP) solvers in dealing with constraint satisfaction problems. In particular, the approach consists of a declarative modeling language based on second-order DLs under Henkin semantics, and a mechanism for transforming second-order DL formulas into a format processable by ASP solvers.

1 Introduction

The goal of Machine Learning (ML) is the design and development of algorithms that allow computers to evolve behaviors based on empirical data [27]. The automation of the *inductive inference* plays a key role in ML algorithms, though other inferences such as abduction and analogy are also considered. Ideally, the ML task is to discover an operational description of a *target function* $f : X \to Y$ which maps elements in the *instance space* X to the values of a set Y. The target function is unknown, meaning that only a set $\mathcal{D}$ (the *training data*) of points of the form $(x, f(x))$ is provided. However, it may be very difficult in general to learn such a description of f perfectly. In fact, ML algorithms are often expected to acquire only some approximation $\hat{f}$ to f by searching a very large space $\mathcal{H}$ of possible hypotheses (the *hypothesis space*) which depend on the representation chosen for f (the *language of hypotheses*). The output approximation is the one that best fits $\mathcal{D}$ according to a *scoring function* $score(f, \mathcal{D})$. It is assumed that any hypothesis $h \in \mathcal{H}$ that approximates f well w.r.t. a large set of training cases will also approximate it well for new unobserved cases. Summing up, given

a hypothesis space $\mathcal{H}$ and a training data set $\mathcal{D}$, ML algorithms are designed to find an approximation $\hat{f}$ of a target function f s.t.:

1. $\hat{f} \in \mathcal{H}$;
2. $\hat{f}(\mathcal{D}) \approx f(\mathcal{D})$; and/or
3. $\hat{f} = argmax_{f \in \mathcal{H}} score(f, \mathcal{D})$.

These notions have been mathematically formalized in computational learning theory within the Probably Approximately Correct (PAC) learning framework [32]. It has been recently stressed that the first two requirements impose constraints on the possible hypotheses, thus defining a *Constraint Satisfaction Problem* (CSP), whereas the third requirement involves the optimization step, thus turning the CSP into an *Optimization Problem* (OP) [9]. We shall refer to the ensemble of constraints and optimization criteria as the model of the learning task. Models are almost by definition declarative and it is useful to distinguish the CSP, which is concerned with finding a solution that satisfies all the constraints in the model, from the OP, where one also must guarantee that the found solution be optimal *w.r.t.* the optimization function. Examples of typical CSPs in the ML context include variants of so-called *Concept Learning* (see Chapt. 2 in [27] for an introduction).

Research in ML has traditionally focussed on designing effective algorithms for solving particular tasks. However, there is an increasing interest in providing the user with a means for specifying what the ML problem in hand actually is rather than letting him struggle to outline how the solution to that problem needs to be computed. This corresponds to a *model+solver*-based approach to ML, in which the user specifies the problem in a *declarative modeling language* and the system automatically transforms such models into a format that can be used by a *solver* to efficiently generate a solution. In this paper, we propose a *model+solver* approach to Concept Learning which combines the efficacy of *Description Logics* (DLs) [1] in conceptual modeling with the efficiency of *Answer Set Programming* (ASP) solvers (see [5] for an overview) in dealing with CSPs. The approach consists of a declarative modeling language based on second-order DLs under Henkin semantics, and a mechanism for transforming second-order DL formulas into a format processable by ASP solvers. This paper completes the work reported in [23]. In particular, it elaborates more on the modeling of one of the variants of the Concept Learning problem discussed in [23] (more precisely, the CSP version of the problem variant called Concept Induction), and provide a substantial contribution to the solver part which was left as future work in [23].

The paper is structured as follows. Section 2 is devoted to preliminaries on DLs, ASP, and Concept Learning. Section 3 introduces a case study from Concept Learning in DLs which is of interest to this paper. Section 4 describes our *model+solver* approach to the case being studied. Section 5 discusses related work. Section 6 summarizes the contributions of the paper and outlines directions of future work.

Table 1. Syntax and semantics of some typical DL constructs.

bottom (resp. top) concept	$\bot$ (resp. $\top$)	$\emptyset$ (resp. $\Delta^{\mathcal{I}}$)
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
individual	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
nominals	$\{a_1, \ldots, a_n\}$	$\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$
concept negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
concept intersection	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
concept union	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
value restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y\ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y\ (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Self concept	$\exists R.Self$	$\{x \in \Delta^{\mathcal{I}} \mid (x,x) \in R^{\mathcal{I}}\}$
qualified	$\leqslant n\ R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in C^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}\} \leq n\}$
number restriction	$\geqslant n\ R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in C^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}\} \geq n\}$
concept inclusion axiom	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
concept equivalence axiom	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
role inclusion axiom	$R_1 \circ \ldots \circ R_n \sqsubseteq S$	$R_1^{\mathcal{I}} \circ \ldots \circ R_n^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
concept assertion	$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$\langle a, b\rangle : R$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
equality assertion	$a \doteq b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
inequality assertion	$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

2 Preliminaries

2.1 Description Logics

DLs are a family of decidable First Order Logic (FOL) fragments that allow for the specification of structured knowledge in terms of classes (*concepts*), instances (*individuals*), and binary relations between instances (*roles*) [1]. Let $\mathsf{N}_{\mathcal{C}}$, $\mathsf{N}_{\mathcal{R}}$, and $\mathsf{N}_{\mathcal{O}}$ be the alphabet of concept names, role names and individual names, respectively. Complex concepts can be defined from atomic concepts and roles by means of constructors. The syntax of some typical DL constructs is reported in Table 1. A DL knowledge base (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a so-called *terminological box* (TBox) $\mathcal{T}$ and a so-called *assertional box* (ABox) $\mathcal{A}$. The TBox is a finite set of *axioms* which represent either is-a relations (denoted with $\sqsubseteq$) or equivalence (denoted with $\equiv$) relations between concepts, whereas the ABox is a finite set of *assertions* (or *facts*) that represent instance-of relations between individuals (resp. couples of individuals) and concepts (resp. roles). DLs provide logical foundations to the W3C *Web Ontology Language* (OWL) [19]. Thus, when a DL-based ontology language is adopted, an ontology is nothing else than a TBox, and a populated ontology corresponds to a whole DL KB (*i.e.*, encom-

passing also an ABox). In particular, $\mathcal{SROIQ}$ [18] is the logical counterpart of OWL 2. [1] A distinguishing feature of $\mathcal{SROIQ}$ is that it admits inverse roles.

The semantics of DLs can be defined directly with set-theoretic formalizations as shown in Table 1 or through a mapping to FOL as shown in [2]. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for a DL KB $\mathcal{K}$ consists of a domain $\Delta^{\mathcal{I}}$ and a mapping function $\cdot^{\mathcal{I}}$. Under the *Unique Names Assumption* (UNA) [31], individuals are mapped to elements of $\Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$. However UNA does not hold by default in DLs. An interpretation $\mathcal{I}$ is a *model* of $\mathcal{K}$ iff it satisfies all axioms and assertions in $\mathcal{T}$ and $\mathcal{A}$. In DLs a KB represents many different interpretations, i.e. all its models. This is coherent with the *Open World Assumption* (OWA) that holds in FOL semantics. A DL KB is *satisfiable* if it has at least one model. An ABox assertion α is a *logical consequence* of a KB $\mathcal{K}$, written $\mathcal{K} \models \alpha$, if all models of $\mathcal{K}$ are also models of α.

The main reasoning task for a DL KB $\mathcal{K}$ is the *consistency check* which tries to prove the satisfiability of $\mathcal{K}$. This check is performed by applying decision procedures mostly based on tableau calculus. The *subsumption check* aims at proving whether a concept is included in another one according to the subsumption relationship. Another well known reasoning service in DLs is *instance check*, *i.e.*, the check of whether an ABox assertion is a logical consequence of a DL KB. A more sophisticated version of instance check, called *instance retrieval*, retrieves, for a DL KB $\mathcal{K}$, all (ABox) individuals that are instances of the given (possibly complex) concept expression C, *i.e.*, all those individuals a such that $\mathcal{K}$ entails that a is an instance of C. All these reasoning tasks support so-called *standard inferences* and can be reduced to the consistency check. Besides the standard ones, additional so-called *non-standard inferences* have been investigated in DL reasoning [21].

When reasoning in DLs, models can be of arbitrary cardinality. In many applications, however, the domain of interest is known to be finite. This is, *e.g.*, a natural assumption in database theory. In *finite model reasoning* [24], models have a finite yet arbitrary, unknown size. Even more interesting from the application viewpoint is the case where the domain has an a priori known cardinality, more precisely, when the domain coincides with the set of named individuals mentioned in the KB. In their proposal of *bounded model reasoning*, Gaggl *et al.* [11] refer to such models as bounded models. Also, they argue that in many applications this modification of the classical DL semantics represents a more intuitive definition of what is considered and expected as model of some KB. In fact, OWL is often "abused" by practicioners as a constraint language for an underlying fixed domain.

2.2 Answer Set Programming

Based on the stable model (answer set) semantics [12], ASP is an alternative logic programming paradigm oriented towards difficult search problems [3]. ASP solvers (see [5] for an overview) are indeed powerful systems especially designed

[1] `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`

to enumerate all solutions. In the following we give a brief overview of the syntax and semantics of disjunctive logic programs in ASP.

Let U be a fixed countable set of (domain) elements, also called *constants*, upon which a total order $\prec$ is defined. An *atom* α is an expression $p(t_1, \ldots, t_n)$, where p is a predicate of arity $n \geq 0$ and each t_i is either a variable or an element from U (*i.e.*, the resulting language is function-free). An atom is *ground* if it is free of variables. B_U denotes the set of all ground atoms over U. A *(disjunctive) rule* r is of the form

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m$$

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, or a count expression of the form $\#count\{l : l_1, \ldots, l_i\} \bowtie u$, where l is an atom and l_j is a literal (*i.e.*, an atom which can be negated or not), $1 \geq j \geq i$, u a non-negative integer, and $\bowtie \in \{\leq, <, =, >, \geq\}$. Moreover, "not" denotes *default negation*. The *head* of r is the set $head(r) = \{a_1, \ldots, a_n\}$ and the *body* of r is $body(r) = \{b_1, \ldots, b_k, notb_{k+1}, \ldots, notb_m\}$. Furthermore, we distinguish between $body^+(r) = \{b_1, \ldots, b_k\}$ and $body^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* if $n \leq 1$ and a *constraint* if $n = 0$. A rule r is *safe* if each variable in r occurs in $body^+(r)$. A rule r is *ground* if no variable occurs in r. A *fact* is a ground rule with $body(r) = \emptyset$ and $|head(r)| = 1$. An *(input) database* is a set of facts. A *program* is a finite set of rules. For a program Π and an input database D, we often write $\Pi(D)$ instead of $D \cup \Pi$. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground).

For any program Π, let U_Π be the set of all constants appearing in Π. $Gr(\Pi)$ is the set of rules $r\sigma$ obtained by applying, to each rule $r \in \Pi$, all possible substitutions σ from the variables in r to elements of U_Π. For count-expressions, $\{l : l_1, \ldots, l_n\}$ denotes the set of all ground instantiations of l, governed through $l_1, \ldots, l_n$. An interpretation $I \subseteq B_U$ satisfies a ground rule r iff $head(r) \cap I = \emptyset$ whenever $body^+(r) \subseteq I$, $body^-(r) \cap I = \emptyset$, and for each contained count-expression, $N \bowtie u$ holds, where N is the cardinality of the set of ground instantiations of l, $N = |\{l | l_1, \ldots, l_n\}|$, for $\bowtie \in \{\leq, <, =, >, \geq\}$ and u a non-negative integer. I satisfies a ground program Π, if each $r \in \Pi$ is satisfied by I. A non-ground rule r (resp., a program Π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\Pi)$). $I \subseteq B_U$ is an answer set of Π iff it is a subset- minimal set satisfying the *Gelfond-Lifschitz reduct* $\Pi^I = \{head(r) \leftarrow body^+(r) | I \cap body^-(r) = \emptyset, r \in Gr(\Pi)\}$. For a program Π, we denote the set of its answer sets by $AS(\Pi)$.

2.3 Concept Learning

Concept Learning deals with inferring the general definition of a category based on members (positive examples) and nonmembers (negative examples) of this category [27]. Here, the target is a Boolean-valued function $f : X \to \{0, 1\}$, *i.e.* a *concept*. When examples of the target concept are available, the resulting ML task is said *supervised*, otherwise it is called *unsupervised*. The positive examples

are those instances with $f(x) = 1$, and negative ones are those with $f(x) = 0$. In Concept Learning, the key inferential mechanism for induction is *generalization as search* through a partially ordered space of inductive hypotheses [26]. Hypotheses may be ordered from the most general ones to the most specific ones. We say that an instance $x \in X$ satisfies a hypothesis $h \in \mathcal{H}$ if and only if $h(x) = 1$. Given two hypotheses h_i and h_j, h_i is more general than or equal to h_j (written $h_i \succeq_g h_j$, where $\succeq_g$ denotes a *generality relation*) if and only if any instance satisfying h_j, also satisfies h_i. Note that it may not be always possible to compare two hypotheses with a generality relation: the instances satisfied by the hypotheses may intersect, and not necessarily be subsumed by one another. The relation $\succeq_g$ defines a *partial order* (*i.e.*, it is reflexive, antisymmetric, and transitive) over the space of hypotheses.

A hypothesis h that correctly classifies all training examples is called *consistent* with these examples. For a consistent hypothesis h it holds that $h(x) = f(x)$ for each instance x. The set of all hypotheses consistent with the training examples is called the *version space* with respect to $\mathcal{H}$ and $\mathcal{D}$. Concept Learning algorithms may use the hypothesis space structure to efficiently search for relevant hypotheses, *e.g.*, they may perform a specific-to-general search through the hypothesis space along one branch of the partial ordering, to find the most specific hypothesis consistent with the training examples. An important issue in Concept Learning is associated with the so-called *inductive bias*, *i.e.* the set of assumptions that the learning algorithm uses for prediction of outputs given previously unseen inputs. These assumptions represent the nature of the target function, so the learning approach implicitly makes assumptions on the correct output for unseen examples. A distinguishing feature of *Inductive Logic Programming* (ILP) [29] with respect to other forms of Concept Learning is the use of prior knowledge of the domain of interest, called *background knowledge* (BK), during the search for hypotheses.

3 The case study

Concept Learning in DLs has been paid increasing attention over the last decade. Notably, algorithms such as [22] have been proposed that follow the *generalization as search* approach by extending the methodological apparatus of ILP to DL languages. In [23], we formally defined three variants of the Concept Learning problem in the DL setting. The variants share the following two features:

1. The background knowledge is in the form of a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and
2. The target theory is a set of DL concept definitions, *i.e.* concept equivalence axioms having an atomic concept in the left-hand side.

but differ in the requirements that an induced concept definition must fulfill in order to be considered as a correct (or valid) solution. The variant we consider in this paper is the supervised one. It is the base for the other variants being introduced in [23]. In the following, the set of all individuals occurring in $\mathcal{A}$ and the set of all individuals occurring in $\mathcal{A}$ that are instance of a given concept C w.r.t. $\mathcal{K}$ are denoted by $\mathsf{Ind}(\mathcal{A})$ and $\mathsf{Retr}_{\mathcal{K}}(C)$, respectively.

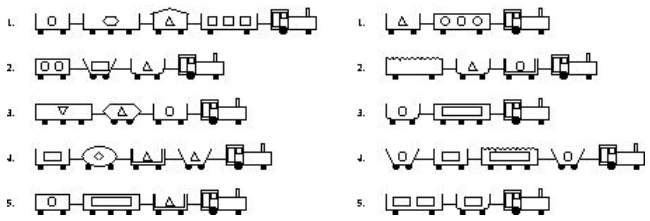

Fig. 1. Michalski's example of eastbound (left) and westbound (right) trains (illustration taken from [25]).

Definition 1 (Concept Induction - CSP version). *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a DL KB. Given:*

- *a (new) target concept name C*
- *a set of positive and negative examples $\mathsf{Ind}_C^+(\mathcal{A}) \cup \mathsf{Ind}_C^-(\mathcal{A}) \subseteq \mathsf{Ind}(\mathcal{A})$ for C*
- *a concept description language $\mathcal{DL}_{\mathcal{H}}$*

the CSP version of the Concept Induction (CI-CSP) *problem is to find a concept definition $C \equiv D$ with $D \in \mathcal{DL}_{\mathcal{H}}$ such that*

Completeness $\mathcal{K} \models (a : D) \quad \forall a \in \mathsf{Ind}_C^+(\mathcal{A})$ *and*
Consistency $\mathcal{K} \models (b : \neg D) \quad \forall b \in \mathsf{Ind}_C^-(\mathcal{A})$

Here, the sets of positive and negative examples are defined as follows

- $\mathsf{Ind}_C^+(\mathcal{A}) = \{a \in \mathsf{Ind}(\mathcal{A}) \mid (a : C) \in \mathcal{A}\} \subseteq \mathsf{Retr}_{\mathcal{K}}(C)$
- $\mathsf{Ind}_C^-(\mathcal{A}) = \{b \in \mathsf{Ind}(\mathcal{A}) \mid (b : \neg C) \in \mathcal{A}\} \subseteq \mathsf{Retr}_{\mathcal{K}}(\neg C)$

These sets can be easily computed by resorting to instance retrieval inference services usually available in DL systems.

Example 1. For illustrative purposes throughout the paper, we choose a very popular learning task in ILP proposed 20 years ago by Ryszard Michalski [25] and illustrated in Figure 1. Here, 10 trains are described, out of which 5 are eastbound and 5 are westbound. The aim of the learning problem is to find the discriminating features between these two classes.

For the purpose of this case study, we have considered an $\mathcal{ALCO}$ ontology, *trains2*, encoding the original Trains data set [2] and distributed with the DL-Learner system. [3] The ontology encompasses 345 logical axioms, 32 classes, 5 object properties and 50 individuals. With reference to *trains2* (which therefore will play the role of $\mathcal{K}$ as in Def. 1), we might want to induce a $\mathcal{SROIQ}$ concept definition for the target concept name $C =$ `EastTrain` from the following positive and negative examples:

[2] `http://archive.ics.uci.edu/ml/datasets/Trains`
[3] `http://dl-learner.org/Projects/DLLearner`

– $\mathsf{Ind}^{+}_{\texttt{EastTrain}}(\mathcal{A}) = \{\texttt{east1}, \ldots, \texttt{east5}\}$
– $\mathsf{Ind}^{-}_{\texttt{EastTrain}}(\mathcal{A}) = \{\texttt{west6}, \ldots, \texttt{west10}\}$

We remind the reader that the examples are chosen from the sets $\mathsf{Retr}_{\mathcal{K}}(\texttt{EastTrain})$ and $\mathsf{Retr}_{\mathcal{K}}(\neg\texttt{EastTrain})$, respectively. Note that the 5 positive examples for `EastTrain` are negative examples for `WestTrain` and viceversa.

4 The approach

The proposed approach consists of a declarative modeling language based on second-order DLs under Henkin semantics (see Sect. 4.1), and a mechanism for transforming second-order DL formulas into a format processable by ASP solvers (see Sect. 4.2). In particular, the transformation is a two-stage process. Second-order DL formulas are instantiated, then encoded as answer set programs.

4.1 Modeling with Second-Order DLs

Let $\mathcal{DL}$ be any DL with syntax $(\mathsf{N}_{\mathcal{C}}, \mathsf{N}_{\mathcal{R}}, \mathsf{N}_{\mathcal{O}})$. In order to write second-order formulas, we introduce a set $\mathsf{N}_{\mathcal{X}} = \{X_0, X_1, X_2, \ldots\}$ of *concept variables*, which we can quantify over. We denote by $\mathcal{DL}_X$ the language of concept terms obtained from $\mathcal{DL}$ by adding $\mathsf{N}_{\mathcal{X}}$.

Definition 2 (Concept term). *A* concept term *in $\mathcal{DL}_X$ is a concept formed according to the specific syntax rules of $\mathcal{DL}$ augmented with the additional rule $C \longrightarrow X$ for $X \in \mathsf{N}_{\mathcal{X}}$.*

Since we are not interested in second-order DLs in themselves, we restrict our language to particular existential second-order formulas of interest to this paper, *i.e.* formulas involving concept subsumptions and concept assertions.

Definition 3 (Second-order concept expression). *Let $a_1, \ldots, a_k \in \mathcal{DL}$ be individuals, $C_1, \ldots, C_m, D_1, \ldots, D_m \in \mathcal{DL}_X$ be concept terms containing concept variables $X_0, \ldots, X_n$. A* concept expression *γ in $\mathcal{DL}_X$ is a conjunction*

$$\begin{array}{c}(C_1 \sqsubseteq D_1) \wedge \ldots \wedge (C_l \sqsubseteq D_l) \wedge (C_{l+1} \not\sqsubseteq D_{l+1}) \wedge \ldots \wedge (C_m \not\sqsubseteq D_m) \wedge \\ (a_1 : D_1) \wedge \ldots \wedge (a_j : D_l) \wedge (a_{j+1} : \neg D_{l+1}) \wedge \ldots \wedge (a_k : \neg D_m)\end{array} \tag{1}$$

of (negated or not) concept subsumptions and concept assertions with $1 \leq l \leq m$ and $1 \leq j \leq k$.

Definition 4 (Second-order formula). *A* formula *ϕ in $\mathcal{DL}_X$ has the form*

$$\exists X.\gamma \tag{2}$$

where γ is a concept expression of the form (1) and X is a concept variable.

We use *General Semantics*, also called Henkin semantics [17], for interpreting concept variables. A nice feature of the Henkin style is that the expressive power of the language actually remains first-order. In such a semantics, variables denoting unary predicates can be interpreted only by *some subsets* among all the ones in the powerset of the domain $2^{\Delta^{\mathcal{I}}}$ - instead, in Standard Semantics a concept variable could be interpreted as *any subset* of $\Delta^{\mathcal{I}}$. Adapting General Semantics to our problem, the structure we consider is exactly composed by the sets interpreting concepts in $\mathcal{DL}$, *i.e.* the interpretation $X^{\mathcal{I}}$ of a concept variable $X \in \mathcal{DL}_X$ must coincide with the interpretation $E^{\mathcal{I}}$ of some concept $E \in \mathcal{DL}$. The interpretations we refer to in the following definition are of this kind.

Definition 5 (Satisfiability of second-order concept expressions). *A concept expression γ of the form (1) is* satisfiable *in $\mathcal{DL}$ iff there exist a concept $E \in \mathcal{DL}$ such that, extending the semantics of $\mathcal{DL}$ for each interpretation $\mathcal{I}$, with: $(X)^{\mathcal{I}} = (E)^{\mathcal{I}}$, it holds that*

1. *for each $j = 1, \ldots, l$, and every $\mathcal{I}$, $(C_j)^{\mathcal{I}} \subseteq (D_j)^{\mathcal{I}}$ and $(a_j)^{\mathcal{I}} \in (D_j)^{\mathcal{I}}$, and*
2. *for each $j = l+1, \ldots, m$, there exists an interpretation $\mathcal{I}$ s.t. $(C_j)^{\mathcal{I}} \not\subseteq (D_j)^{\mathcal{I}}$ and $(a_j)^{\mathcal{I}} \in (\neg D_j)^{\mathcal{I}}$*

Otherwise, γ is said to be unsatisfiable *in $\mathcal{DL}$.*

Definition 6 (Solution for a second-order concept expression). *Let γ be a concept expression of the form (1). If γ is satisfiable in $\mathcal{DL}$, then E is a* solution *for γ.*

Definition 7 (Satisfiability of second-order formulas). *A formula ϕ of the form (2) is* true *in $\mathcal{DL}$ if there exist at least a solution for γ, otherwise it is* false.

The fragment of Second-Order DLs just introduced can be used as a declarative modeling language for Concept Learning problems in DLs. Following Def. 1, we assume that $\mathsf{Ind}_C^+(\mathcal{A}) = \{a_1, \ldots, a_m\}$ and $\mathsf{Ind}_C^-(\mathcal{A}) = \{b_1, \ldots, b_n\}$. A concept $D \in \mathcal{DL}_{\mathcal{H}}$ is a correct concept definition for the target concept name C w.r.t. $\mathsf{Ind}_C^+(\mathcal{A})$ and $\mathsf{Ind}_C^-(\mathcal{A})$ iff it is a solution for the following second-order concept expression:

$$\gamma_{\mathsf{CI\text{-}CSP}} := (a_1 : X) \sqcap \ldots \sqcap (a_m : X) \sqcap (b_1 : \neg X) \sqcap \ldots \sqcap (b_n : \neg X) \tag{3}$$

that is, iff D can be an assignment for the concept variable X. The CI-CSP problem can be modeled with the following second-order formula

$$\phi_{\mathsf{CI\text{-}CSP}} := \exists X.\gamma_{\mathsf{CI\text{-}CSP}} \tag{4}$$

The solvability of a CI-CSP problem is therefore based on the satisfiability of the second-order formula being used for modeling the problem.

Definition 8 (Solvability of CI-CSP problems). *A CI-CSP problem $\boldsymbol{P}$ is* solvable *if $\phi_{\mathsf{CI\text{-}CSP}}$ is true in $\mathcal{DL}_{\mathcal{H}}$. Otherwise, the problem is* not solvable. *If D is a solution for $\gamma_{\mathsf{CI\text{-}CSP}}$, then $C \equiv D$ is a solution for $\boldsymbol{P}$.*

Example 2. According to (3), the intended CI-CSP problem of Example 1 corresponds to the following second-order concept expression $\gamma_{\texttt{EastTrain}}$:

$$(\texttt{east1} : X) \wedge \ldots \wedge (\texttt{east5} : X) \wedge (\texttt{west6} : \neg X) \wedge \ldots \wedge (\texttt{west10} : \neg X) \quad (5)$$

The problem is then solvable if the following second-order formula:

$$\phi_{\texttt{EastTrain}} := \exists X.\gamma_{\texttt{EastTrain}} \quad (6)$$

is true in $\mathcal{SROIQ}$, *i.e.*, if there exists a solution to $\gamma_{\texttt{EastTrain}}$ in $\mathcal{SROIQ}$.

4.2 Solving with ASP

In order to solve the problems modeled with the second-order concept expressions introduced in the previous section we need mechanisms for generating and evaluating candidate solutions.

How to generate candidate solutions The CI-CSP problem statement reported in Def. 1 mentions a concept description language $\mathcal{DL}_{\mathcal{H}}$ among its inputs. It is the language of hypotheses and allows for the generation of concept definitions in any $\mathcal{DL}$ according to some declarative bias. It can be considered as a generative grammar.

The concept expressions generated from $\mathcal{DL}_{\mathcal{H}}$ can be organized according to the concept subsumption relation $\sqsubseteq$. Note that $\sqsubseteq$ is a reflexive and transitive binary relation, *i.e.* a quasi-order. Thus, $(\mathcal{DL}_{\mathcal{H}}, \sqsubseteq)$ is a quasi-ordered set of $\mathcal{DL}$ concept definitions which defines a search space to be traversed either top-down or bottom-up by means of suitable refinement operators according to the *generalization as search* approach in Mitchell's vision.

Definition 9 (Refinement operator in DLs). *Given a quasi-ordered search space* $(\mathcal{DL}_{\mathcal{H}}, \sqsubseteq)$

- *a* downward refinement operator *is a mapping* $\rho : \mathcal{DL}_{\mathcal{H}} \to 2^{\mathcal{DL}_{\mathcal{H}}}$ *such that*
$$\forall C \in \mathcal{DL}_{\mathcal{H}} \quad \rho(C) \subseteq \{D \in \mathcal{DL}_{\mathcal{H}} \mid D \sqsubseteq C\}$$
- *an* upward refinement operator *is a mapping* $\delta : \mathcal{DL}_{\mathcal{H}} \to 2^{\mathcal{DL}_{\mathcal{H}}}$ *such that*
$$\forall C \in \mathcal{DL}_{\mathcal{H}} \quad \delta(C) \subseteq \{D \in \mathcal{DL}_{\mathcal{H}} \mid C \sqsubseteq D\}$$

Note that there is an infinite number of generalizations and specializations in a given $(\mathcal{DL}, \sqsubseteq)$. Usually one tries to define refinement operators that can traverse efficiently the hypothesis space in pursuit of one of the correct definitions (w.r.t. the examples that have been provided).

Example 3. Let us assume that the language of hypotheses allows for the generation of $\mathcal{SROIQ}$ concept expressions starting from the atomic concept and role names occurring in *trains2* (except, of course, for the target concept name). Among the concepts (instantiations of X) satisfying $\gamma_{\texttt{EastTrain}}$, there is

$$\geqslant 3\, \texttt{hasCar}.(\neg \texttt{JaggedCar}) \quad (7)$$

which describes the set of trains composed by at least three cars that are not jagged. It provides a correct concept definition for `EastTrain` w.r.t. the given examples, *i.e.*, the following concept equivalence axiom

$$\texttt{EastTrain} \equiv \geqslant 3\, \texttt{hasCar}.(\neg \texttt{JaggedCar}) \tag{8}$$

is a solution for the CI-CSP problem in hand.

How to evaluate candidate solutions The choice of the solver is a critical aspect in any *model+solver* approach. In our case the use of a second-order modeling language does not necessarily imply the use of a second-order solver. Indeed, the Henkin semantics paves the way to the use of first-order solvers. Once instantiated, concept expressions of the kind (3) are just first-order DL conjunctive queries. However, the CSP nature of the problem in hand should not be neglected. This led us to assume the bounded model semantics for the instantiated concept expressions instead of the classical one.

As already mentioned in Section 2.1, Gaggl *et al.* [11] modify the modelhood condition by restricting the domain to a finite set of bounded size, induced by the named individuals occurring in the given OWL ontology (denoted as $\mathsf{N}_{\mathcal{O}}(\mathcal{K})$). This means that, under the bounded model semantics, there is a one-to-one correspondence between interpretations and sets of ground facts, if one assumes the set of domain elements fixed and known. In other words, ABoxes can used as representations of models.

Definition 10 (Bounded model semantics [11]). *Let $\mathcal{K}$ be a $\mathcal{SROIQ}$ KB. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is said to be* individual-bounded *w.r.t. $\mathcal{K}$, if all of the following holds:*

1. $\Delta^{\mathcal{I}} = \{a | a \in \mathsf{N}_{\mathcal{O}}(\mathcal{K})\}$,
2. *for each individual* $a \in \mathsf{N}_{\mathcal{O}}(\mathcal{K})\}$, $a^{\mathcal{I}} = a$.

Accordingly, an interpretation $\mathcal{I}$ is an (individual-)bounded model *of $\mathcal{K}$, if $\mathcal{I}$ is an individual-bounded interpretation w.r.t. $\mathcal{K}$ and $\mathcal{I} \models \mathcal{K}$ holds.*

Also, $\mathcal{K}$ is called bm-satisfiable *if it has a bounded model.*

We say that $\mathcal{K}$ bm-entails *an axiom α (written $K \models_{bm} \alpha$) if every bounded model of $\mathcal{K}$ is also a model of α.*

The benefits of bounded model semantics are manifolded.

First, this non-classical semantics is computationally advantageous. Indeed, while reasoning in OWL under the classical semantics is N2ExpTime-complete [20], reasoning under the bounded model semantics is merely NP-complete [11].

Second, an arbitrary $\mathcal{SROIQ}$ KB $\mathcal{K}$ can be encoded into an answer set program $\Pi(\mathcal{K})$, such that the set of answer sets $AS(\Pi(\mathcal{K}))$ coincides with the set of bounded models of the given KB [11]. The rules for transforming $\mathcal{SROIQ}$ concept expressions into ASP are reported in Table 2. Here, $\mathsf{O_a}$ is a new concept name unique for the individual a. Also, $ar(r, X, Y)$ is defined as follows:

$$ar(r, X, Y) := \begin{cases} \mathsf{R}(X, Y) & \text{if } R \text{ is an atomic role} \\ \mathsf{S}(Y, X) & \text{if } R \text{ is an inverse role and } R = S^- \end{cases}$$

Table 2. Translation of $\mathcal{SROIQ}$ concept expressions into ASP (adapted from [11]).

C	$trans(C)$
A	$not\ \mathsf{A}(X)$
$\neg A$	$\mathsf{A}(X)$
$\{a\}$	$\{not\ \mathsf{O_a}(X)\},\ \{\mathsf{O_a}(X)\}$
$\forall R.A$	$\{not\ \mathsf{A}(Y_A), ar(r, X, Y_A)\}$
$\forall R.(\neg A)$	$\{ar(r, X, Y_A), \mathsf{A}(Y_A)\}$
$\exists R.Self$	$not\ ar(r, X, X)$
$\neg\exists R.Self$	$ar(r, X, X)$
$\geqslant n\ R.A$	$\#count\{ar(r, X, Y_A) : \mathsf{A}(Y_A)\} < n$
$\geqslant n\ R.(\neg A)$	$\#count\{ar(r, X, Y_A) : not\ \mathsf{A}(Y_A)\} < n$
$\leqslant n\ R.A$	$\#count\{ar(r, X, Y_A) : \mathsf{A}(Y_A)\} > n$
$\leqslant n\ R.(\neg A)$	$\#count\{ar(r, X, Y_A) : not\ \mathsf{A}(Y_A)\} > n$

The translation into ASP requires a KB to be in the so-called *normalized form* (see [28] for details) which can be obtained however by an easy syntactic transformation. The encoding turns out to be a more effective alternative to the axiomatization, since existing OWL reasoners struggle on bounded model reasoning, due to the heavy combinatorics involved.

Last, but not least, we particularly emphasize OWL as modeling language for typical CSPs.

Example 4. Let $\gamma'_{\texttt{EastTrain}}$ be the first-order concept expression obtained by instantiating the unique second-order variable in $\gamma_{\texttt{EastTrain}}$ with the concept (7) generated by some refinement operator for $\mathcal{SROIQ}$. It can be encoded in ASP under bounded model semantics. The resulting ASP program can be then checked for satisfiability by any ASP solver.

5 Related Work

The *model+solver* approach to ML/DM has been promoted by De Raedt *et al.* [9,10] and successfully applied to one of the most popular DM tasks: Constraint-based pattern mining [30,14,15,16]. Along this line, Guns *et al.* [13] introduce MiningZinc, a general framework for constraint-based pattern mining. It consists of two key components: a language component and a toolchain component. The language allows for high-level and natural modeling of mining problems, such that MiningZinc models closely resemble definitions found in the data mining literature. It is inspired by the Zinc family of languages and systems and supports user-defined constraints and optimization criteria. The toolchain allows for finding solutions to the models. It ensures the solver independence of the language and supports both standard constraint solvers and specialized data mining systems. Automatic model transformations enable the efficient use of different solvers and systems. The combination of both components allows one to rapidly

model constraint-based mining problems and execute these with a wide variety of methods.

Bruynooghe *et al.* [4] suggest that predicate logic can be useful as a modeling language and show how to model and solve ML and DM problems with IDP3. The core of IDP3 is a finite model generator that supports FOL enriched with types, inductive definitions, aggregates and partial functions. It offers its users a modeling language that is a slight extension of predicate logic and allows them to solve a wide range of search problems. Apart from a small introductory example, applications are selected from problems that arose within ML/DM research. These research areas have recently shown a strong interest in declarative modeling and constraint solving as opposed to algorithmic approaches. The paper illustrates that the IDP3 system can be a valuable tool for researchers with such an interest.

Colucci *et al.* [7] have proposed a unified framework for non-standard reasoning services in DLs. The framework is based on the use of second-order sentences in DLs [6]. It applies to so-called *constructive inferences*, *i.e.*, those non-standard inferences that deal with finding (or constructing) a concept. More precisely, it provides a unifying definition model for all those constructive reasoning tasks which rely on specific optimality criteria to build up the objective concept. Indeed, constructive reasoning tasks can be divided into two main categories: Tasks for which we just need to compute a concept (or a set of concepts) and those for which we need to find a concept (or a set of concepts) according to some minimality/maximality criteria. In the first case, we have a set of solutions while in the second one we also have a set of sub-optimal solutions to the main problem. For instance, the set of sub-optimal solutions in the Least Common Subsumer (LCS) problem is represented by the common subsumers. In [7], Colucci *et al.* provide also a sound and complete procedure for solving constructive reasoning problems in DLs, which combines a tableaux calculus for DLs with rules for the substitution of concept variables in second-order concept expressions. However, the procedure does not terminate in all cases, since some of the above problems are known to be undecidable. More recently, Colucci and Donini [8] have presented a modular Prolog prototype system aimed at proving the feasibility of their unified framework for non-standard reasoning in DLs. The prototype supports only some constructive reasoning problems, *e.g.* LCS, and two simple DLs ($\mathcal{EL}$ and $\mathcal{ALN}$).

6 Summary and Directions of Future Work

In this paper we have carried on the work reported in [23] by studying in more depth the case of Concept Induction, the basic case of Concept Learning in DLs which can be naturally reformulated as a CSP. In particular, we have proposed a *model+solver* approach to Concept Induction which consists of a declarative modeling language based on second-order DLs under Henkin semantics, and a mechanism for transforming second-order DL formulas into a format processable by ASP solvers. The transformation is possible under bounded model semantics,

a non-standard model-theoretic semantics for DLs which has been recently proposed in order to correctly address CSPs in OWL.

In the future, we plan to implement the approach. Also we intend to investigate how to express optimality criteria such as the information gain function within the second-order concept expressions.

Acknowledgements The author would like to thank Sebastian Rudolph and Sarah Alice Gaggl for the fruitful discussions about the bounded model semantics for OWL.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications (2nd ed.). Cambridge University Press (2007)
2. Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial Intelligence 82(1–2), 353–367 (1996)
3. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011), http://doi.acm.org/10.1145/2043174.2043195
4. Bruynooghe, M., Blockeel, H., Bogaerts, B., de Cat, B., De Pooter, S., Jansen, J., Labarre, A., Ramon, J., Denecker, M., Verwer, S.: Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with *IDP3*. Theory and Practice of Logic Programming 15(6), 783–817 (2015), http://dx.doi.org/10.1017/S147106841400009X
5. Calimeri, F., Ianni, G., Ricca, F.: The third open answer set programming competition. Theory and Practice of Logic Programming 14(1), 117–135 (2014), http://dx.doi.org/10.1017/S1471068412000105
6. Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F.M., Ragone, A.: Second-order description logics: Semantics, motivation, and a calculus. In: Haarslev, V., Toman, D., Weddell, G.E. (eds.) Proceedings of the 23rd International Workshop on Description Logics (DL 2010), Waterloo, Ontario, Canada, May 4-7, 2010. CEUR Workshop Proceedings, vol. 573. CEUR-WS.org (2010)
7. Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F.M., Ragone, A.: A unified framework for non-standard reasoning services in description logics. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings. Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 479–484. IOS Press (2010)
8. Colucci, S., Donini, F.M.: Inverting subsumption for constructive reasoning. In: Kazakov, Y., Lembo, D., Wolter, F. (eds.) Proceedings of the 2012 International Workshop on Description Logics, DL-2012, Rome, Italy, June 7-10, 2012. CEUR Workshop Proceedings, vol. 846. CEUR-WS.org (2012)
9. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for data mining and machine learning. In: Fox, M., Poole, D. (eds.) Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010. AAAI Press (2010)
10. De Raedt, L., Nijssen, S., O'Sullivan, B., Van Hentenryck, P.: Constraint programming meets machine learning and data mining (Dagstuhl seminar 11201). Dagstuhl Reports 1(5), 61–83 (2011)

11. Gaggl, S.A., Rudolph, S., Schweizer, L.: Bound your models! How to make OWL an ASP modeling language. CoRR abs/1511.00924 (2015), http://arxiv.org/abs/1511.00924
12. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9(3/4), 365–386 (1991)
13. Guns, T., Dries, A., Tack, G., Nijssen, S., De Raedt, L.: MiningZinc: A modeling language for constraint-based mining. In: Rossi, F. (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013. IJCAI/AAAI (2013)
14. Guns, T., Nijssen, S., De Raedt, L.: Evaluating pattern set mining strategies in a constraint programming framework. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) Advances in Knowledge Discovery and Data Mining - 15th Pacific-Asia Conference, PAKDD 2011, Shenzhen, China, May 24-27, 2011, Proceedings, Part II. Lecture Notes in Computer Science, vol. 6635, pp. 382–394. Springer (2011)
15. Guns, T., Nijssen, S., De Raedt, L.: Itemset mining: A constraint programming perspective. Artificial Intelligence 175(12-13), 1951–1983 (2011)
16. Guns, T., Nijssen, S., De Raedt, L.: k-pattern set mining under constraints. IEEE Trans. Knowl. Data Eng. 25(2), 402–418 (2013)
17. Henkin, L.: Completeness in the theory of types. Journal of Symbolic Logic 15(2), 81–91 (1950)
18. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006. pp. 57–67. AAAI Press (2006)
19. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics 1(1), 7–26 (2003)
20. Kazakov, Y.: $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008. pp. 274–284 (2008), http://www.aaai.org/Library/KR/2008/kr08-027.php
21. Küsters, R.: Non-Standard Inferences in Description Logics, Lecture Notes in Computer Science, vol. 2100. Springer (2001)
22. Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. Machine Learning 78(1-2), 203–250 (2010)
23. Lisi, F.A.: A declarative modeling language for concept learning in description logics. In: Riguzzi, F., Zelezny, F. (eds.) Inductive Logic Programming, 22nd International Conference, ILP 2012, Dubrovnik, Croatia, September 17-19, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7842. Springer Berlin Heidelberg (2013)
24. Lutz, C., Sattler, U., Tendera, L.: The complexity of finite model reasoning in description logics. Information and Computation 199(1-2), 132–171 (2005), http://dx.doi.org/10.1016/j.ic.2004.11.002
25. Michalski, R.: Pattern recognition as a rule-guided inductive inference. IEEE transactions on Pattern Analysis and Machine Intelligence 2(4), 349–361 (1980)
26. Mitchell, T.M.: Generalization as search. Artificial Intelligence 18, 203–226 (1982)
27. Mitchell, T.M.: Machine Learning. McGraw Hill (1997)
28. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. Artif. Intell. Research 36, 165–228 (2009), http://dx.doi.org/10.1613/jair.2811

29. Muggleton, S.H.: Inductive logic programming. In: Arikawa, S., Goto, S., Ohsuga, S., Yokomori, T. (eds.) Proceedings of the 1st Conference on Algorithmic Learning Theory. Springer/Ohmsma (1990)
30. Nijssen, S., Guns, T., De Raedt, L.: Correlated itemset mining in ROC space: a constraint programming approach. In: Elder IV, J.F., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (eds.) Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009. pp. 647–656. ACM (2009)
31. Reiter, R.: Equality and domain closure in first order databases. Journal of ACM 27, 235–249 (1980)
32. Valiant, L.: A theory of the learnable. Communications of the ACM 27(11), 1134–1142 (1984)