Mach Learn (2010) 78: 203–250
DOI 10.1007/s10994-009-5146-2

Concept learning in description logics using refinement operators

Jens Lehmann · Pascal Hitzler

Received: 15 September 2007 / Revised: 19 August 2009 / Accepted: 19 August 2009 /
Published online: 16 September 2009

Abstract With the advent of the Semantic Web, description logics have become one of the most prominent paradigms for knowledge representation and reasoning. Progress in research and applications, however, is constrained by the lack of well-structured knowledge bases consisting of a sophisticated schema and instance data adhering to this schema. It is paramount that suitable automated methods for their acquisition, maintenance, and evolution will be developed. In this paper, we provide a learning algorithm based on refinement operators for the description logic ALCQ including support for concrete roles. We develop the algorithm from thorough theoretical foundations by identifying possible abstract property combinations which refinement operators for description logics can have. Using these investigations as a basis, we derive a practically useful complete and proper refinement operator. The operator is then cast into a learning algorithm and evaluated using our implementation *DL-Learner*. The results of the evaluation show that our approach is superior to other learning approaches on description logics, and is competitive with established ILP systems.

Keywords Description logics · Refinement operators · Inductive logic programming · Semantic web · OWL · Structured machine learning

1 Introduction and motivation

With the advent of the Semantic Web and Semantic Technologies, ontologies are becoming one of the most prominent paradigms for knowledge representation and reasoning. However,

Editor: Hendrik Blockeel.

J. Lehmann (✉)
Department of Computer Science, Universität Leipzig, Johannisgasse 26, 04103 Leipzig, Germany
e-mail: lehmann@informatik.uni-leipzig.de

P. Hitzler
Kno.e.sis Center, Wright State University, 3640 Colonel Glenn Hwy, Dayton, OH 45435, USA
e-mail: pascal@pascal-hitzler.de

recent progress in the field faces a lack of well-structured ontologies with large amounts of instance data due to the fact that engineering such ontologies constitutes a considerable investment of resources. Nowadays, knowledge bases often provide large amounts of instance data without sophisticated schemata. Methods for automated schema acquisition and maintenance are therefore being sought (see e.g. Buitelaar et al. 2007).

In 2004, the World Wide Web Consortium (W3C) recommended the Web Ontology Language OWL[1] as a standard for modelling ontologies on the Web. In the meantime, many studies and applications using OWL have been reported in research, many of which go beyond Internet usage and employ the power of ontological modelling in other fields like biology, medicine, software engineering, knowledge management, and cognitive systems (Staab and Studer 2004; Davies et al. 2006; Hitzler et al. 2009).

In essence, OWL coincides with the description logic $\mathcal{SHOIN}$(D) and is likely to be extended to $\mathcal{SROIQ}$(D) (Horrocks et al. 2006) in OWL 2. Description logics (DLs) in general are fragments of first order logic. In order to leverage machine-learning approaches for the acquisition of OWL ontologies, it is required to develop methods and tools for learning in description logics. To date, only few investigations have been carried out on this topic, which can be attributed to the fact that description logics (DLs) have only recently become a major paradigm in knowledge representation and reasoning applications.

In this paper, we show that methods from Inductive Logic Programming (ILP) are applicable to learning in description logic knowledge bases. (Throughout the paper, we use the terms knowledge base and ontology synonymously.) We are motivated by the success of ILP and believe that similar success can be achieved for DLs. We believe that our results provide foundations for the acquisition of ontologies, in particular in cases when extensional information (facts, instance data) is easily available, while corresponding intensional information (schema) is missing or not expressive enough to allow powerful reasoning over the ontology in a useful way. Such situations often occur when extracting knowledge from different sources, e.g. databases[2] and wikis (Auer et al. 2008), or in collaborative knowledge engineering scenarios.

The developed system is, of course, not restricted to ontology engineering and can handle other learning problems. Indeed, it lends itself to generic use in machine learning in the same way as ILP systems do. The main difference, however, is the employed knowledge representation paradigm: ILP traditionally uses logic programs for knowledge representation while our work rests on description logics. This distinction is crucial when considering Semantic Web applications as target use cases for our approach, as such applications hinge centrally on the chosen knowledge representation format for knowledge interchange and integration.[3] As such, our work can be understood as a broadening of the scope of research and applications based on ILP methods. This is particularly important since the number of description logic based systems can be expected to be increasing rapidly in the near future.[4]

A central part in many ILP approaches are so-called *refinement operators* which are used to traverse the search space, and such an ILP-based approach often hinges on the definition of a suitable operator. Theoretical investigations on ILP refinement operators have

[1] http://www.w3.org/2004/OWL/. See also http://www.w3.org/2007/OWL/ for the currently ongoing revision of the standard.

[2] E.g. http://triplify.org.

[3] We will say more about this in Sect. 8.

[4] As a case in point, see the W3C LinkingOpenData Project, http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData.

identified desirable properties for them to have, which impact on their performance. These properties thus provide general guidelines for the definition of suitable operators, which are not restricted to a logic programming setting, but can be carried over to DLs. It turns out, however, that for expressive DLs there are strong theoretical limitations on the properties a refinement operator can have. A corresponding general analysis therefore provides a clear understanding of the difficulties inherent in a learning setting, and also allows to derive directions for researching suitable operators.

We provide both this theoretical analysis and the derivation—including experimental validation—of a refinement operator suitable for concept learning in description logics. The theoretical analysis is provided in Sect. 4, with Theorem 2 being the central result of this part. A concrete refinement operator is given in Definition 13 based on a definition from the beginning of Sect. 5. Experimental evaluations are provided in Sect. 7.

In more detail, the structure of the paper is as follows. In Sect. 2, we give a brief introduction to description logics. Section 3 formally describes the learning problem. Refinement operators and their properties are introduced. Section 4 contains the theoretical analysis of possible property combinations, culminating in Theorem 2, which summarises the theoretical part. Indeed, we fully analyse all combinations of a set of interesting properties of refinement operators known from the literature. This means that we show which combinations of properties are possible, i.e. for which combinations a refinement operator with these properties exists. To the best of our knowledge, such a complete analysis has not been done before. The results provide a foundation for further investigations into practical refinement operators, independent from the rest of this paper. In Sect. 5, we then propose a concrete refinement operator. Based on the theoretical investigations in Sect. 4, we argue that such an operator should be complete and proper, and thus we first define a complete operator, which is later extended into a complete and proper operator. This operator supports the description logic $\mathcal{ALC}$ and is then extended to an operator for $\mathcal{ALCQ}$ with support for concrete roles. In Sect. 6, we present a learning algorithm based on this operator, and in Sect. 7 we evaluate this algorithm by means of our implementation, *DL-Learner*. The evaluation shows that our approach is competitive on several benchmark problems. In Sect. 8, we discuss in detail the advantages and disadvantages of our approach using description logics compared to traditional logic programming based methods. In Sect. 9, we discuss related work, in particular the relation to refinement operators in the area of traditional Inductive Logic Programming. Finally, in Sect. 10 we summarise our work and draw conclusions.

Note This paper is a substantially extended and revised merge of the two papers (Lehmann and Hitzler 2007a, 2007b), which have been awarded the Best Student Paper Award at the 17th International Conference on Inductive Logic Programming, ILP 2007, Corvallis, OR, USA. Changes going beyond the merge of these papers are as follows:

- extension of the used refinement operator with usage of domain and range restrictions, disjoint concepts, support for cardinality restrictions, role hierarchies, boolean and double concrete roles
- support and description of a new heuristic
- extension of the evaluation part
- inclusion of all proofs (adapted to the algorithm extension where necessary)
- inclusion of more discussion and related work

Additionally, we published our implementation DL-Learner[5] and all learning examples/benchmarks as open source software.

[5]Homepage: http://dl-learner.org, download: http://sf.net/projects/dl-learner.

2 Description logics

Description logics is the name of a family of knowledge representation (KR) formalisms. They emerged from earlier KR formalisms like semantic networks and frames. Their origin lies in the work of Brachman on structured inheritance networks (Brachman 1978). Since then, description logics have enjoyed increasing popularity. They can essentially be understood as fragments of first-order predicate logic. They have less expressive power, but usually decidable inference problems and a user-friendly variable free syntax.

Description logics represent knowledge in terms of *objects*, *concepts*, and *roles*. Concepts formally describe notions in an application domain, e.g. we could define the concept of being a father as "a man having a child" ($\texttt{Father} \equiv \texttt{Man} \sqcap \exists \texttt{hasChild}.\top$ in DL notation). Objects are members of concepts in the application domain and roles are binary relations between objects. Objects correspond to constants, concepts to unary predicates, and roles to binary predicates in first-order logic.

In description logic systems information is stored in a *knowledge base*. It is divided in two parts: *TBox* and *ABox*. The ABox contains *assertions* about objects. It relates objects to concepts and other objects via roles. The TBox describes the *terminology* by relating concepts and roles. (For some expressive description logics this clear separation does not exist.)

As mentioned before, DLs are a family of KR formalisms. We will introduce the $\mathcal{ALC}$ description logic as a prototypical example. It should be noted that $\mathcal{ALC}$ is a proper fragment of OWL (Horrocks et al. 2003) and is generally considered to be a prototypical description logic for research investigations.

$\mathcal{ALC}$ stands for *attributive language with complement*. It allows to construct complex concepts from simpler ones using various language constructs. The next definition shows how such concepts can be built.

Definition 1 (Syntax of $\mathcal{ALC}$ concepts) Let N_R be a set of *role names* and N_C be a set of *concept names* ($N_R \cap N_C = \emptyset$). The elements of N_C are also called *atomic concepts*. The set of $\mathcal{ALC}$ concepts is inductively defined as follows:

1. Each atomic concept is an $\mathcal{ALC}$ concept.
2. If C and D are $\mathcal{ALC}$ concepts and $r \in N_R$ a role, then the following are also $\mathcal{ALC}$ concepts:
   - $\top$ (top), $\bot$ (bottom)
   - $C \sqcup D$ (disjunction), $C \sqcap D$ (conjunction), $\neg C$ (negation)
   - $\forall r.C$ (value/universal restriction), $\exists r.C$ (existential restriction)

For some of the results in this paper we will refer to sublanguages of $\mathcal{ALC}$. Such sublanguages are defined by disallowing the use of a selection of the language constructs disjunction, conjunction, negation, and universal or existential restriction. When doing this, we will always assume that $\top$ and $\bot$ are in the language. We will explicitly refer in some places to the language $\mathcal{AL}$, the concepts of which are inductively defined as follows: $\top$, $\bot$, $\exists r.\top$, A, $\neg A$ with $A \in N_C$, $r \in N_R$ are $\mathcal{AL}$ concepts. If C and D are $\mathcal{AL}$ concepts, then $C \sqcap D$ is an $\mathcal{AL}$ concept. If C is an $\mathcal{AL}$ concept and r a role, then $\forall r.C$ is an $\mathcal{AL}$ concept.

Only for one result (Lemma 2 and the results based on it), we will have to refer explicitly to description logics which are more expressive than $\mathcal{ALC}$. More precisely, we will refer to $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$ as the two languages underlying OWL and OWL 2, respectively.

Table 1 $\mathcal{ALC}$ syntax and semantics

Construct	Syntax	Semantics
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
abstract role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
exists restriction	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid \exists b.(a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$
universal restriction	$\forall r.C$	$(\forall r.C)^{\mathcal{I}} = \{a \mid \forall b.(a,b) \in r^{\mathcal{I}}$ implies $b \in C^{\mathcal{I}}\}$

Since we need them formally only for this single proof, we refrain from giving the formal definitions, and instead refer the reader to Horrocks et al. (2006), Baader et al. (2007), Hitzler et al. (2009) for details.

The semantics of $\mathcal{ALC}$ concepts is defined by means of interpretations. See the following definition and Table 1 listing all $\mathcal{ALC}$ concept constructors.

Definition 2 (Interpretation) An *interpretation* $\mathcal{I}$ consists of a non-empty *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$, which assigns to each $A \in N_C$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to each $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

In the most general case, *terminological axioms* are of the form $C \sqsubseteq D$ or $C \equiv D$, where C and D are concepts. The former axioms are called *inclusions* and the latter *equivalences*. An equivalence whose left hand side is an atomic concept is a *concept definition*. In some languages with low expressivity, like $\mathcal{AL}$, terminological axioms are restricted to definitions. We can define the semantics of terminological axioms in a straightforward way. An interpretation $\mathcal{I}$ satisfies an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and it satisfies the equivalence $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. $\mathcal{I}$ satisfies a set of terminological axioms iff it satisfies all axioms in the set. An interpretation, which satisfies a (set of) terminological axiom(s) is called a *model* of this (set of) axiom(s). Two (sets of) axioms are *equivalent* if they have the same models. A finite set $\mathcal{T}$ of terminological axioms is called a (*general*) *TBox*. Let N_I be the set of object names (disjoint with N_R and N_C). An *assertion* has the form $C(a)$ (*concept assertion*) or $r(a,b)$ (*role assertion*), where a, b are object names, C is a concept, and r is a role. An *ABox* $\mathcal{A}$ is a finite set of assertions.

Objects are also called individuals. To allow interpreting ABoxes we extend the definition of an interpretation. In addition to mapping concepts to subsets of our domain and roles to binary relations, an interpretation has to assign to each individual name $a \in N_I$ an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a model of an ABox $\mathcal{A}$ (written $\mathcal{I} \models \mathcal{A}$) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all $C(a) \in \mathcal{A}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for all $r(a,b) \in \mathcal{A}$. An interpretation $\mathcal{I}$ is a model of a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ (written $\mathcal{I} \models \mathcal{K}$) iff it is a model of $\mathcal{T}$ and $\mathcal{A}$.

A concept is in *negation normal form* iff negation only occurs in front of concept names. The *length* of a concept is defined in a straightforward way, namely as the sum of the numbers of concept names, role names, quantifiers, and connective symbols occurring in the concept. The *depth* of a concept is the maximal number of nested concept constructors. The

role depth of a concept is the maximal number of nested roles. A *subconcept* of a concept C is a concept syntactically contained in C. For brevity we sometimes omit brackets. In this case, constructors involving quantifiers have higher priority, e.g. $\exists r.\top \sqcap A$ means $(\exists r.\top) \sqcap A$. In several proofs in the paper we use a convenient abbreviated notation to denote $\forall r$ *chains* and $\exists r$ *chains*:

$$\forall r^n = \underbrace{\forall r. \ldots . \forall r}_{n\text{-times}} \qquad \exists r^n = \underbrace{\exists r. \ldots . \exists r}_{n\text{-times}}$$

As we have described, a knowledge base can be used to represent the information we have about an application domain. Besides this *explicit* knowledge, we can also deduce *implicit* knowledge from a knowledge base. It is the aim of *inference algorithms* to extract such implicit knowledge. There are some standard reasoning tasks in description logics, which we will briefly describe.

In *terminological reasoning* we reason about concepts. The standard problems are *satisfiability* and *subsumption*. Intuitively, satisfiability determines if a concept can be satisfied, i.e. it is free of contradictions. Subsumption of two concepts detects whether one of the concepts is more general than the other.

Definition 3 (Satisfiability) Let C be a concept and $\mathcal{T}$ a TBox. C is *satisfiable* iff there is an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. C is *satisfiable with respect to* $\mathcal{T}$ iff there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.

Definition 4 (Subsumption, equivalence) Let C, D be concepts and $\mathcal{T}$ a TBox. C *is subsumed by* D, denoted by $C \sqsubseteq D$, iff for all models $\mathcal{I}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. C *is subsumed by* D *with respect to* $\mathcal{T}$, denoted by $C \sqsubseteq_{\mathcal{T}} D$, iff for all models $\mathcal{I}$ of $\mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

C *is equivalent to* D (*with respect to* $\mathcal{T}$), denoted by $C \equiv D$ ($C \equiv_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and $D \sqsubseteq C$ ($D \sqsubseteq_{\mathcal{T}} C$).

C *is strictly subsumed by* D *(with respect to* $\mathcal{T}$*)*, denoted by $C \sqsubset D$ ($C \sqsubset_{\mathcal{T}} D$), iff $C \sqsubseteq D$ ($C \sqsubseteq_{\mathcal{T}} D$) and not $C \equiv D$ ($C \equiv_{\mathcal{T}} D$).

Subsumption allows to build a hierarchy of atomic concepts, commonly called the *subsumption hierarchy*. Analogously, for more expressive description logics *role hierarchies* can be inferred.

In *assertional reasoning* we reason about objects. As one relevant task for learning in DLs, the *instance check problem* is to find out whether an object is an instance of a concept, i.e. belongs to it.

Definition 5 (Instance) Let $\mathcal{A}$ be an ABox, $\mathcal{T}$ a TBox, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a knowledge base, C a concept, and $a \in N_I$ an object. a *is an instance of* C *with respect to* $\mathcal{A}$, denoted by $\mathcal{A} \models C(a)$, iff in all models $\mathcal{I}$ of $\mathcal{A}$ we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$. a *is an instance of* C *with respect to* $\mathcal{K}$, denoted by $\mathcal{K} \models C(a)$, iff in all models $\mathcal{I}$ of $\mathcal{K}$ we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

To denote that a is not an instance of C with respect to $\mathcal{A}$ ($\mathcal{K}$) we write $\mathcal{A} \not\models C(a)$ ($\mathcal{K} \not\models C(a)$).

We use the same notation for sets S of assertions of the form $C(a)$, e.g. $\mathcal{K} \models S$ means that every element in S follows from $\mathcal{K}$. $\mathcal{K} \not\models S$ means that no element in S follows from $\mathcal{K}$.

3 Learning in description logics using refinement operators

In this section, we briefly describe the learning problem in description logics. The process of learning concepts in logics, i.e. of finding a logical description for given instances, is also called *inductive reasoning*. In a general setting this means that we have a logical formulation of background knowledge and some observations. We are then looking for ways to extend the background knowledge such that we can explain the observations, i.e. they can be deduced from the modified knowledge. More formally, we are given background knowledge B, positive examples E^+, negative examples E^- and want to find a hypothesis H such that from H together with B the positive examples follow and the negative examples do not follow. It is not required that the same logical formalism is used for background knowledge, examples, and hypothesis, but often this is the case.

Definition 6 (Learning problem in description logics) Let a concept name `Target`, a knowledge base $\mathcal{K}$ (not containing `Target`), and sets E^+ and E^- with elements of the form $\texttt{Target}(a)$ ($a \in N_I$) be given. The learning problem is to find a concept C such that `Target` does not occur in C and for $\mathcal{K}' = \mathcal{K} \cup \{\texttt{Target} \equiv C\}$ we have $\mathcal{K}' \models E^+$ and $\mathcal{K}' \not\models E^-$. Such a concept C is called *correct*.

If an example a is instance of a concept C, we say that C *covers* a.

In the learning problem definition we made an acyclicity restriction, e.g. we do not learn recursive equivalences. The main reason for this is performance. If we know that the equivalence we learn is not cyclic, we can perform instance checks to test example coverage without adding the definition to the knowledge base first. Moreover, cyclic equivalences are less common in ontologies compared to logic programs. Clearly, if the need for them arises, the approaches can be extended to include this case.

By Occam's razor (Blumer et al. 1990; Domingos 1998) simple and more comprehensible solutions of the learning problem are to be preferred over more complex ones. We measure simplicity as the syntactic length of a concept as defined in Sect. 2 and will later bias our algorithm towards shorter concepts.

The goal of learning is to find a correct concept with respect to the examples. This can be seen as a search process in the space of concepts. A natural idea is to impose an ordering on this search space and use operators to traverse it. This idea is well-known in Inductive Logic Programming (Nienhuys-Cheng and de Wolf 1997), where refinement operators are widely used to find hypotheses. Intuitively, downward (upward) refinement operators construct specialisations (generalisations) of hypotheses.

Definition 7 (Refinement operator) A *quasi-ordering* is a reflexive and transitive relation. In a quasi-ordered space $(S, \preceq)$ a *downward (upward) refinement operator* ρ is a mapping from S to 2^S, such that for all $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$ ($C \preceq C'$). C' is called a *specialisation* (*generalisation*) of C.

This idea can be used for searching in the space of concepts. As ordering we can use subsumption. (Note that the subsumption relation $\sqsubseteq$ is a quasi-ordering.) If a concept C subsumes a concept D ($D \sqsubseteq C$), then C will cover all examples which are covered by D. This makes subsumption a suitable order for searching in concepts. We analyse refinement operators for concepts with respect to subsumption and a description language $\mathcal{L}$, and in the sequel we will call such operators $\mathcal{L}$ *refinement operators*. We also introduce the commonly used notions of refinement chains as well as downward and upward covers.

Definition 8 ($\mathcal{L}$ refinement operator) Let $\mathcal{L}$ be a description language. A refinement operator in the quasi-ordered space $(\mathcal{L}, \sqsubseteq)$ is called an $\mathcal{L}$ *refinement operator*.

Definition 9 (Refinement chain) A *refinement chain* of an $\mathcal{L}$ refinement operator ρ of length n from a concept C to a concept D is a finite sequence $C_0, C_1, \ldots, C_n$ of concepts, such that $C = C_0, C_1 \in \rho(C_0), C_2 \in \rho(C_1), \ldots, C_n \in \rho(C_{n-1}), D = C_n$. This refinement chain *goes through* E iff there is an i $(1 \leq i \leq n)$ such that $E = C_i$. We say that D can be reached from C by ρ if there exists a refinement chain from C to D. $\rho^*(C)$ denotes the set of all concepts, which can be reached from C by ρ. $\rho^m(C)$ denotes the set of all concepts, which can be reached from C by a refinement chain of ρ of length m.

Definition 10 (Downward and upward cover) A concept C is a *downward cover* of a concept D iff $C \sqsubset D$ and there does not exist a concept E with $C \sqsubset E \sqsubset D$. A concept C is an *upward cover* of a concept D iff $D \sqsubset C$ and there does not exist a concept E with $D \sqsubset E \sqsubset C$.

Instead of $D \in \rho(C)$, we will often write $C \rightsquigarrow_\rho D$. If the used operator is clear from the context, it is usually omitted, i.e. we write $C \rightsquigarrow D$.

We introduce the notion of weak equality of concepts, which is similar to syntactic equality of concepts, but takes into account that the order of elements in conjunctions and disjunctions is not important. We say that the concepts C and D are *weakly* (*syntactically*) *equal*, denoted by $C \simeq D$ iff they are equal up to permutation of arguments of conjunction and disjunction. Two sets S_1 and S_2 of concepts are weakly equal iff for all $C_1 \in S_1$ there is a $C_1' \in S_2$ such that $C_1 \simeq C_1'$ and vice versa. Weak equality of concepts is coarser than syntactic equality and finer than equivalence (viewing the equivalence, equality, and weak equality of concepts as equivalence classes).

Refinement operators can have certain properties, which can be used to evaluate their usefulness for learning hypotheses.

Definition 11 (Properties of DL refinement operators) An $\mathcal{L}$ refinement operator ρ is called

- (*locally*) *finite* iff $\rho(C)$ is finite for all concepts C.
- *redundant* iff there exists a refinement chain from a concept C to a concept D, which does not go through some concept E and a refinement chain from C to a concept weakly equal[6] to D, which does go through E.
- *proper* iff for all concepts C and D, $D \in \rho(C)$ implies $C \not\equiv D$.
- *ideal* iff it is finite, complete (see below), and proper.

An $\mathcal{L}$ downward refinement operator ρ is called

- *complete* iff for all concepts C, D with $C \sqsubset D$ we can reach a concept E with $E \equiv C$ from D by ρ.
- *weakly complete* iff for all concepts $C \sqsubset \top$ we can reach a concept E with $E \equiv C$ from $\top$ by ρ.
- *minimal* iff for all C, $\rho(C)$ contains only downward covers and all its elements are incomparable with respect to $\sqsubseteq$.

The corresponding notions for upward refinement operators are defined dually.

[6] We use weak equality instead of syntactic equality, because only avoiding syntactic equality while still being able to reach many weakly equal concepts from a given concept can still waste significant computational resources. All results also hold if we consider syntactic equality, see Remark 3.

4 Analysing the properties of refinement operators

In this section, we analyse the properties of refinement operators in description logics. The need for such an analysis was already expressed in Fanizzi et al. (2004), Esposito et al. (2004). In particular, we are interested in finding out which desired properties can be combined in a refinement operator and which properties are impossible to combine. This is interesting for two reasons: The first one is that this gives us a good impression of how hard (or easy) it is to learn concepts. The second reason is that this can also serve as a practical guide for designing refinement operators. Knowing the theoretical limits allows the designer of a refinement operator to focus on achieving the best possible properties. We will indeed follow this approach in Sect. 5, where we will define—and later evaluate—a concrete operator based on our theoretical investigations.

Refinement operators for Description Logics have been constructed for $\mathcal{ALER}$ in Badea and Nienhuys-Cheng (2000), and in Esposito et al. (2004), Iannone and Palmisano (2005), Iannone et al. (2007) for $\mathcal{ALC}$. In particular, Badea and Nienhuys-Cheng (2000) also showed some properties of $\mathcal{ALER}$ refinement operators. However, a full theoretical treatment of their properties has not been done to the best of our knowledge (not even for a specific language). Therefore, all propositions in this section are new unless explicitly mentioned otherwise.

As a first property we will briefly analyse minimality of $\mathcal{L}$ refinement operators, in particular the existence of upward and downward covers in $\mathcal{ALC}$. It is not immediately obvious that e.g. downward covers exist in $\mathcal{ALC}$, because it could be the case that for all concepts C and D with $C \sqsubset D$ one can always construct a concept E with $C \sqsubset E \sqsubset D$. However, the next proposition shows that downward covers do exist.

Proposition 1 (Existence of covers in $\mathcal{ALC}$) *Downward* (*upward*) *covers of* $\top$ ($\bot$) *exist in* $\mathcal{ALC}$.

Proof Let $N_R = \{r\}$ and $N_C = \{A\}$.

Assume, we have an interpretation $\mathcal{I}$ and an object a such that $a^{\mathcal{I}} \notin A^{\mathcal{I}}$ and there is no b with $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ (*). We define a set S as follows:

- $\top \in S$
- $\neg A \in S$
- for all concepts C, $\forall r.C \in S$
- if $C_1 \in S$ and $C_2 \in S$, then $C_1 \sqcap C_2 \in S$
- if $C_1 \in S$ or $C_2 \in S$, then $C_1 \sqcup C_2 \in S$

By structural induction on $\mathcal{ALC}$ concept constructors (see Table 1), it can be shown that $C \in S$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all C in negation normal form.

Using this observation as prerequisite, we show that $C = \exists r.\top \sqcup A$ is a downward cover of $\top$. By contradiction, we assume that there is a concept D with $C \sqsubset D \sqsubset \top$ in negation normal form.

Since $C \sqsubset D$, there are $\mathcal{I}_1$ and a_1 such that $a_1^{\mathcal{I}_1} \notin C^{\mathcal{I}_1}$ and $a_1^{\mathcal{I}_1} \in D^{\mathcal{I}_1}$. Analogously, due to $D \sqsubset \top$, there are $\mathcal{I}_2$ and a_2 such that $a_2^{\mathcal{I}_2} \notin D^{\mathcal{I}_2}$. By $C \sqsubset D$, this implies $a_2^{\mathcal{I}_2} \notin C^{\mathcal{I}_2}$. In summary (**):

I_1: $a_1^{\mathcal{I}_1} \notin C^{\mathcal{I}_1}, a_1^{\mathcal{I}_1} \in D^{\mathcal{I}_1}$
I_2: $a_2^{\mathcal{I}_2} \notin C^{\mathcal{I}_2}, a_2^{\mathcal{I}_2} \notin D^{\mathcal{I}_2}$

We can deduce:

$$\begin{aligned} & a_1^{\mathcal{I}_1} \notin C^{\mathcal{I}_1} \\ \iff \quad & a_1^{\mathcal{I}_1} \notin (A \sqcup \exists r.\top)^{\mathcal{I}_1} \\ \iff \quad & a_1^{\mathcal{I}_1} \notin A^{\mathcal{I}_1} \text{ and } a_1^{\mathcal{I}_1} \notin (\exists r.\top)^{\mathcal{I}_1} \\ \iff \quad & a_1^{\mathcal{I}_1} \notin A^{\mathcal{I}_1} \text{ and there is no } b \text{ with } (a^{\mathcal{I}_1}, b^{\mathcal{I}_1}) \in r^{\mathcal{I}_1} \end{aligned}$$

The same can be done for a_2 and $\mathcal{I}_2$. In both cases assumption (*) is satisfied.

If $D \in S$, then $a_1^{\mathcal{I}_1} \in D^{\mathcal{I}_1}$ and $a_2^{\mathcal{I}_2} \in D^{\mathcal{I}_2}$. Otherwise if $D \notin S$, then $a_1^{\mathcal{I}_1} \notin D^{\mathcal{I}_1}$ and $a_2^{\mathcal{I}_2} \notin D^{\mathcal{I}_2}$. Both cases contradict (**).

Upward covers can be handled analogously, i.e. $\forall r.\bot \sqcap A$ is an upwardcover of $\bot$. □

The idea in the proof of Proposition 1 can be extended to situations with more than one role and concept name. In this case we obtain the following concept as a downward cover of $\top$ (we do not prove this explicitly, because we do not use this result later on):

$$\bigsqcup_{r \in N_R} \exists r.\top \sqcup \bigsqcup_{A \in N_C} A$$

The result shows that non-trivial minimal operators, i.e. operators which do not map every concept to the empty set, can be constructed. However, minimality of refinement steps is not a directly desired goal in general. Minimal operators are in some languages more likely to lead to overfitting, because they may not produce sufficient generalisation/specialisation leaps, e.g. the cover above provides almost no specialisation compared to $\top$. This problem is particularly significant in languages which are closed under boolean operations, i.e. in $\mathcal{ALC}$ and more expressive languages.

Indeed, the following result suggests that—unlike for logic programs—minimality may not play a central role for DL refinement operators, as it is incompatible with weak completeness. In Badea and Nienhuys-Cheng (2000), a weaker result was already claimed to hold, but not proven. We formulate our result for the description logic $\mathcal{AL}$. A corresponding result for other description logics than $\mathcal{AL}$ has not been shown yet, but the non-existence of such operators even for weak description logics suggests that similar problems arise for more expressive ones.

Proposition 2 (Minimality and weak completeness) *There exists no minimal and weakly complete $\mathcal{AL}$ downward refinement operator.*

Proof Let $N_R = \{r\}$, and $N_C = \emptyset$. In the following, let $\mathrm{rd}(C)$ denote the role depth of a concept C.

For a proof by contradiction, we assume we have a minimal and weakly complete $\mathcal{AL}$ downward refinement operator ρ. This implies that there is a refinement step $C \rightsquigarrow_\rho D$, such that C does not have a subconcept equivalent to $\bot$ and D does have a subconcept equivalent to $\bot$. Otherwise no concept equivalent to $\bot$ would be reachable from $\top$. We prove the proposition by first simplifying C and D and then defining a concept E with $D \sqsubset E \sqsubset C$.

We next cast C and D into a normal form. For this, we replace the subconcepts in D equivalent to $\bot$ by $\bot$. Furthermore, we apply the following equivalence preserving rewrite rules exhaustively to C and D:

$$C \sqcap \bot \to \bot \quad \text{and} \quad \bot \sqcap C \to \bot$$

$$C \sqcap \top \to C \quad \text{and} \quad \top \sqcap C \to C$$

$$\forall r.\top \to \top$$

$$\forall r.(C_1 \sqcap C_2) \to \forall r.C_1 \sqcap \forall r.C_2$$

We pick a normal form obtained this way (note that applying the rules to an arbitrary concept can lead to different results depending on the order of application) and call the resulting concepts $C'(\equiv C)$ and $D'(\equiv D)$.

Due to the syntax of $\mathcal{AL}$, $N_C = \emptyset$, and the rewriting rules, we have that C' is either $\top$ or of the following form, where the s_i, for $i \in \{1, \ldots, b\}$, are non-negative integers:

$$C' = \prod_{i=1}^{b} \forall r^{s_i}.\exists r.\top \tag{1}$$

We define a new concept E:

$$E = C' \sqcap \forall r^n.\exists r.\top \quad \text{with } n = \max(\mathrm{rd}(C), \mathrm{rd}(D)) + 1$$

We complete the proof by showing that $D' \sqsubset E \sqsubset C'$, which contradicts the minimality of ρ. The central idea behind the contradiction is that the $\forall r$-chain in E is strictly longer than all of those occurring in C' or D'. This guarantees both the containment of E in C' and the containment of D' in E. The details are as follows.

1. To show $E \sqsubset C'$, first note that $E = C' \sqcap \forall r^n.\exists r.\top \sqsubseteq C'$ is obvious, so it remains to show that E and C' are not equivalent. To do this, we define $\mathcal{I}$ and a such that $a^{\mathcal{I}} \in C'$ and $a^{\mathcal{I}} \notin E$.

   Let $\mathcal{I}$ be defined by $r^{\mathcal{I}} = \{(a_i, a_{i+1}) \mid a_0 = a, 0 \le i < n\}$, which we can depict as

   $$a = a_0 \xrightarrow{r} a_1 \xrightarrow{r} \cdots \xrightarrow{r} a_n$$

   – Indeed $a^{\mathcal{I}} \in C'$ holds: For $C' = \top$ this holds trivially. If $C' \neq \top$, then C' is a conjunction of concepts which are of the form $\forall r^m.\exists r.\top$ with $m < n$, i.e. we have $a^{\mathcal{I}} \in (\forall r^m.\exists r.\top)^{\mathcal{I}}$ for all $m < n$. Note that the statement $\forall r^m.\exists r.\top$ informally means that objects reachable via a path of length m along r need to have a successor. Hence $a^{\mathcal{I}} \in C'$ holds due to the form we have established in (1) for C'.
   – $a^{\mathcal{I}} \notin E$ holds because a_n does not have an r-filler.

2. To show $D' \sqsubset E$, first note that D' contains a $\bot$ symbol. This means that D' is of the form $D' = D'_1 \sqcap \cdots \sqcap D'_p$ $(p \ge 1)$ where there exists some $j \in \{1, \ldots, p\}$ s.t. D'_j is of the form $\forall r^t.\bot$, for some $t \ge 0$. Now define $F = C' \sqcap D'_j$. Note that[7] $D' \equiv C' \sqcap D' \sqsubseteq C' \sqcap D'_j = F$.

   We first verify $D' \sqsubseteq E$. Note that for all $k, l \ge 0$ we have

   $$\forall r^k.\bot \sqsubseteq \forall r^k.\forall r^l.\exists r.\top$$

   and since $t < n$ we obtain

   $$D' \sqsubseteq F = C' \sqcap \forall r^t.\bot \sqsubseteq C' \sqcap \forall r^n.\exists r.\top = E$$

[7] We have $D' \equiv C' \sqcap D'$, because $D \equiv D'$ is a downward refinement of $C \equiv C'$, i.e. $D' \sqsubseteq C'$. $C' \sqcap D' \sqsubseteq C' \sqcap D'_j$ holds because of $D' \sqsubseteq D'_j$.

It remains to show that $D' \not\equiv E$. We do this by proving $F \not\equiv E$ using the interpretation $\mathcal{I}$ with $r^{\mathcal{I}} = \{(a, a)\}$.

We have $a^{\mathcal{I}} \in (\forall r^z.\exists r.\top)^{\mathcal{I}}$ for all $z \geq 0$. Hence, we have $a^{\mathcal{I}} \in C'^{\mathcal{I}}$ and also $a^{\mathcal{I}} \in E^{\mathcal{I}}$.

We have $a^{\mathcal{I}} \notin F^{\mathcal{I}}$, because $a^{\mathcal{I}} \notin (\forall r^t.\bot)^{\mathcal{I}}$—note that $\forall r^t.\bot$ states that objects reachable via a path of length $t-1$ must not have a successor.

Hence, $a^{\mathcal{I}} \in E^{\mathcal{I}}$ and $a^{\mathcal{I}} \notin F^{\mathcal{I}}$, which proves $F \not\equiv E$. □

Please note that although we looked at the special case of $N_R = \{r\}$ and $N_C = \emptyset$ in the proof, similar arguments could be used for arbitrary N_R and N_C.

In the sequel, we analyse desired properties of $\mathcal{L}$ refinement operators: completeness, properness, finiteness, and non-redundancy. As we have just shown that minimality does not play a central role when investigating expressive DLs, we omit it from further investigations. We show several positive and negative results, which together yield a full analysis of these properties.

Proposition 3 (Complete and finite refinement operators) *Let $\mathcal{L}$ be a description language which allows conjunction. Then there exists a complete and finite $\mathcal{L}$ refinement operator.*

Proof Consider the downward refinement operator ρ defined by

$$\rho(C) = \{C \sqcap \top\} \cup \{D \mid |D| \leq (\text{number of } \top \text{ occurrences in } C) \text{ and } D \sqsubset C\}$$

where $|D|$ stands for the number of symbols in D. The operator can do one of two things:

- add a $\top$ symbol
- generate the set of all concepts up to a certain length, which are subsumed by C

The operator is finite, because the set of all concepts up to a given length is finite (and the singleton set $\{C \sqcap \top\}$ is finite).

The operator is complete, because given a concept C we can reach an arbitrary concept D with $D \sqsubset C$. This is obvious, because we only need to add $\top$-symbols until there are $|D|$ occurrences of $\top$. Within the next step we can then be sure to reach D. □

For upward refinement operators we can use an analogous operator ϕ, which is also complete and finite:

$$\phi(C) = \{C \sqcap \top\} \cup \{D \mid |D| \leq (\text{number of } \top \text{ occurrences in } C) \text{ and } C \sqsubset D\}$$

Remark 1 It has been claimed in Badea and Nienhuys-Cheng (2000) that complete and finite $\mathcal{ALER}$ refinement operators do not exist. However, this is refuted by Proposition 3.

Of course, it is obvious that the operator used to prove Proposition 3 is not useful in practice, since it merely generates concepts without paying attention to efficiency. However, here we are interested in theoretical limits of refinement operators, so it is a valid method to consider impractical operators. It is indeed difficult to design a good complete and finite refinement operator. The reason is that finiteness can only be achieved by using non-proper refinement steps (for our operator this was done by adding $\top$ symbols). We will now prove this, i.e. show that it is impossible to define a complete, finite, and proper refinement operator. Such operators are known as ideal and their non-existence indicates that learning concepts in sufficiently expressive description logics is hard.

Lemma 1 *Let* $C = \neg\exists r^n.\top \sqcup \exists r^{n+1}.\top$. *There is no concept* D *with* $C \sqsubseteq D \sqsubset \top$ *and role depth smaller* n *in* $\mathcal{ALC}$ *or* $\mathcal{ALCQ}$.

Proof Note that C is not equivalent to $\top$. For the interpretation $\mathcal{I}$ with $r^{\mathcal{I}} = \{(a_i, a_{i+1}) \mid 0 \leq i < n\}$, illustrated by $a_0 \xrightarrow{r^{\mathcal{I}}} a_1 \xrightarrow{r^{\mathcal{I}}} \cdots \xrightarrow{r^{\mathcal{I}}} a_n$, we have $a_0^{\mathcal{I}} \notin C^{\mathcal{I}}$.

By contradiction, we assume that such a concept D exists.

We can view an interpretation $\mathcal{I}$ as a directed graph with respect to r in a straightforward way: The set of nodes is $\{b^{\mathcal{I}} \mid b \in N_I\}$ and the edges are $\{(b, c) \mid (b, c) \in r^{\mathcal{I}}\}$. We define $\mathrm{lp}_{\mathcal{I},a}$ as the length of the longest paths in the graph of $\mathcal{I}$ starting from $a^{\mathcal{I}}$ ($a \in N_I$). If such a path is infinite, we set $\mathrm{lp}_{\mathcal{I},a} = \infty$.

Let $\mathcal{I}$ be an arbitrary interpretation and a an arbitrary object. If $\mathrm{lp}_{\mathcal{I},a} > n$, we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$ because of $a^{\mathcal{I}} \in (\exists r^{n+1}.\top)^{\mathcal{I}}$ and $\exists r^{n+1}.\top \sqsubseteq C$. If $\mathrm{lp}_{\mathcal{I},a} < n$, we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$, because of $a^{\mathcal{I}} \in (\neg\exists r^n.\top)^{\mathcal{I}}$ and $\neg\exists r^n.\top \sqsubseteq C$. If $\mathrm{lp}_{\mathcal{I},a} = n$, we have $a^{\mathcal{I}} \notin C^{\mathcal{I}}$ because of $a^{\mathcal{I}} \notin (\exists r^{n+1}.\top)^{\mathcal{I}}$ and $a^{\mathcal{I}} \notin (\neg\exists r^n.\top)^{\mathcal{I}}$. So we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$ iff $\mathrm{lp}_{\mathcal{I},a} \neq n$.

Due to $D \not\equiv \top$, we know that $\neg D$ is satisfiable. $\mathcal{ALC}$, $\mathcal{ALCQ}$, and other description logics have the tree model property, i.e. any satisfiable concept has a model, where the graph we defined is tree shaped (Baader et al. 2007). In particular, this means that if there is a path in the model graph between two arbitrary objects, then this path is unique. So without loss of generality, we can assume that the graphs of the considered models are tree shaped. Let $\mathcal{I}$ be a tree shaped model of $\neg D$ and a be an object such that $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$. Note that because of $C \sqsubseteq D$, we know $a^{\mathcal{I}} \notin C^{\mathcal{I}}$.

We know that $\mathrm{lp}_{\mathcal{I},a} = n$ in $\mathcal{I}$, otherwise $a^{\mathcal{I}} \in C^{\mathcal{I}}$ as we have shown above. Let $a^{\mathcal{I}} = a_0 \xrightarrow{r^{\mathcal{I}}} a_1 \xrightarrow{r^{\mathcal{I}}} \cdots \xrightarrow{r^{\mathcal{I}}} a_n$ be one of the longest paths in the graph of $\mathcal{I}$ starting in a. We create a new interpretation $\mathcal{I}'$ from $\mathcal{I}$ by adding a new object a_{m+1} and changing $r^{\mathcal{I}}$ to $r^{\mathcal{I}'} = r^{\mathcal{I}} \cup \{(a_m, a_{m+1})\}$.

The following concept constructors involving roles are included in the mentioned languages: $\forall r$, $\exists r$, $\leq m\, r$, $\geq m\, r$. Semantically, all of those refer to role fillers (see Table 1), i.e. to neighbours in the interpretation graph defined above. Since the role depth of D is smaller than n and there is exactly one path in the graph of I' from $a^{\mathcal{I}}$ to a_{m+1}, we can deduce $a^{\mathcal{I}'} \in (\neg D)^{\mathcal{I}'}$ from $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$ (the addition of a_{m+1} does not influence whether $a^{\mathcal{I}'} \in (\neg D)^{\mathcal{I}'}$ or not). However, $\mathcal{I}'$ has $\mathrm{lp}_{\mathcal{I},a} = n+1$, because we added a_{m+1}. Thus, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ as shown above and hence $a^{\mathcal{I}} \in D^{\mathcal{I}}$, which contradicts $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$. □

Lemma 2 *Let*

$$\begin{aligned} C = {} & \neg\exists r^n.\top \sqcup \exists r^{n+1}.\top \\ & \sqcup \geq 2r.\top \sqcup \exists r.\geq 2r.\top \sqcup \cdots \sqcup \exists r^{n-1}.\geq 2r.\top \\ & \sqcup \exists r^{-1}.\top \sqcup \exists r.\geq 2r^{-1}.\top \sqcup \exists r^2.\geq 2r^{-1}.\top \sqcup \cdots \sqcup \exists r^n.\geq 2r^{-1}.\top \end{aligned}$$

There is no concept D *with* $C \sqsubseteq D \sqsubset \top$ *and role depth smaller* n *in* $\mathcal{SHOIN}$ *or* $\mathcal{SROIQ}$.

Proof We use the same notions as in Lemma 1. C is not equivalent to $\top$ by the same example as in the previous lemma. Again, we assume by contradiction that such a concept D exists. As before, there are a and $\mathcal{I}$ with $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$ (and thus $a^{\mathcal{I}} \notin C^{\mathcal{I}}$). The difference is that $\mathcal{SROIQ}$ and $\mathcal{SHOIN}$ do not have the tree model property.

Since $\neg\exists r^n.\top \sqcup \exists r^{n+1}.\top \sqsubseteq C$, we know that $\mathrm{lp}_{\mathcal{I},a} = n$ by the observations made in Lemma 1. Let $a^{\mathcal{I}} = a_0 \xrightarrow{r^{\mathcal{I}}} a_1 \xrightarrow{r^{\mathcal{I}}} \cdots \xrightarrow{r^{\mathcal{I}}} a_n$ be the longest path in the graph of $\mathcal{I}$ starting in a. Using $a^{\mathcal{I}} \notin C^{\mathcal{I}}$, we can make some observations concerning the graph of $\mathcal{I}$:

– a_0 has no incoming edge (due to $\exists r^{-1}.\top$)
– a_1 to a_n have exactly one incoming edge (due to $\exists r^i. \geq 2\, r^{-1}.\top$ for $1 \leq i \leq n$)
– a_n does not have an outgoing edge (due to $\mathrm{lp}_{\mathcal{I},a} = n$)
– a_0 to a_{n-1} have exactly one outgoing edge (due to $\exists r^i. \geq 2\, r.\top$ for $0 \leq i \leq n-1$)

This means that there is a unique longest path, which forms a connected component of the interpretation graph. The concept constructors involving roles in $\mathcal{SHOIN}$ and $\mathcal{SROIQ}$ are $\forall r$, $\forall r^{-1}$, $\exists r$, $\exists r^{-1}$, $\leq m\, r$, $\leq m\, r^{-1}$, $\geq m\, r$, $\geq m\, r^{-1}$, $\exists r$.Self, all of which refer to direct neighbours (via incoming and outgoing edges) of an object in the graph.

Again, we can construct an interpretation $\mathcal{I}'$ by adding a new object a_{m+1} to the longest path. Since the role depth of D is smaller than n, we can deduce $a^{\mathcal{I}'} \in (\neg D)^{\mathcal{I}'}$ from $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$. However, $\mathcal{I}'$ has $\mathrm{lp}_{\mathcal{I},a} = n+1$, because we added a_{m+1}. Thus, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $a^{\mathcal{I}} \in D^{\mathcal{I}}$, which contradicts $a^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$. □

Proposition 4 (Ideal refinement operators) *There does not exist any ideal $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{SHOIN}$, or $\mathcal{SROIQ}$ downward refinement operator.*

Proof By contradiction, we assume that there exists an ideal downward refinement operator ρ. We further assume that there is a role $r \in N_R$. Let $\rho(\top) = T = \{C_1, \ldots, C_n\}$ be the set of refinements of the $\top$ concept. Due to finiteness of ρ, T has to be finite. Let n be a natural number larger than the maximum of the role depths of concepts in T. Due to Lemmas 1 and 2, we know that there does not exist a more general concept than C (where C is as defined in the corresponding lemma depending on the language we consider), which is not equal to $\top$, with a role depth smaller than n. We have also shown that $C \not\equiv \top$.

Hence $C_1, \ldots, C_n$ do not subsume C (the properness of ρ implies that $C_1, \ldots, C_n$ are not equivalent to $\top$), so C cannot be reached from any of these concepts by applying ρ. Thus, C cannot be reached from $\top$ and ρ is incomplete. □

We limited the proof to a set of interesting languages. However, it generalizes to other languages with only minor adaptations.

Proposition 5 (Complete and proper refinement operators) *Let $\mathcal{L}$ be any description language. Then there exists a complete and proper $\mathcal{L}$ refinement operator.*

Proof To prove this, we can use $\rho(C) = \{D \mid D \sqsubset C\}$ as downward refinement operator, which is obviously complete and proper. For upward refinement we can analogously consider $\rho(C) = \{D \mid C \sqsubset D\}$. □

We have shown that the combination of completeness and properness is possible. Propositions 3, 4, and 5 state that for complete refinement operators, which are usually desirable, one has to sacrifice properness or finiteness. We will now look at non-redundancy.

Proposition 6 (Complete, non-redundant refinement operators) *Let $\mathcal{L}$ be a description language which contains $\mathcal{AL}$. Then there exists a complete and non-redundant $\mathcal{L}$ refinement operator.*

Proof We prove the result by showing that complete operators can be transformed to complete and non-redundant operators. Note that in the following, we will use the role r to create concepts with a certain depth. If N_R does not contain any role, the desired effect can also be

achieved by using conjunctions or disjunctions of $\top$ and $\bot$, but this would render the proof less readable.

We will use the fact that the set of concepts in $\mathcal{L}$ is countably infinite. The countability already follows from the fact that there is just a finite number of concepts with a given length. Hence, we can divide the set of all concepts in finite subsets, where each subset contains all concepts of the same length. We can then start enumerating concepts starting with the subset of concepts of length 1, then length 2 etc. Thus, there is a bijective function $f : \mathcal{L} \mapsto \mathbf{N}$, which assigns a different number to each concept in $\mathcal{L}$. We denote the inverse function mapping numbers to concepts by f^{inv}.

We modify a given complete operator ρ defined as $\rho(C) = S_C$, where S_C is defined as a maximal subset of $\{D \mid D \sqsubset C\}$ with $D_1, D_2 \in S_C \Rightarrow D_1 \not\simeq D_2$. ρ is complete, since given a concept C all more special concepts up to weak equivalence can be reached in one refinement step.

We change ρ in the following way: For all concepts C, $\rho(C)$ is modified by changing any element $D \in \rho(C)$ to

$$D \sqcap \forall r.(\underbrace{\top \sqcap \cdots \sqcap \top}_{f(C) \text{ times}})$$

We claim that the resulting operator, which we denote by ρ' is complete and non-redundant.

The completeness of ρ' follows from the completeness of ρ, since the construct we have added does not change the semantics (it is equivalent to $\top$).

To prove non-redundancy, we will first define a refinement operator ρ^{inv}, which maps conjunctions, where the last element is of the form $\forall r.(\top \sqcap \cdots \sqcap \top)$, to a single concept:

$$\rho^{\text{inv}} = \begin{cases} \{f^{\text{inv}}(n)\} & \text{if } C \text{ is of the form } C \sqcap \forall r.\underbrace{(\top \sqcap \cdots \sqcap \top)}_{n \text{ times}} \\ \emptyset & \text{otherwise} \end{cases}$$

We can see that $D \in \rho'(C)$ implies $\rho^{\text{inv}}(D) = \{C\}$, so ρ^{inv} allows to invert a refinement step of ρ'.

By contradiction, we assume ρ' is redundant. Then, there needs to be a concept C and concepts D_1, D_2 with $D_1 \simeq D_2$, such that there is a refinement chain from C to D_1 and a different refinement chain from C to D_2. Since we know that D_1 and D_2 are refinements of ρ', they have to be conjunctions, where the last element is of the form $\forall r.(\top \sqcap \cdots \sqcap \top)$ and equal in D_1 and D_2. This means the previous element in both refinement chains is $\rho^{\text{inv}}(D_1) = \rho^{\text{inv}}(D_2)$ and therefore all elements except the last ones (D_1, D_2) in both chains are uniquely determined. Since for all C, $\rho(C)$ does not contain weakly equal concepts, we get that D_1 and D_2 have to be equal. Therefore, both refinement chains are equal, which is a contradiction.

We can establish the same result for upward refinement by using ρ with $\rho(C) = S_C$, where S_C is defined as a maximal subset of $\{D \mid C \sqsubset D\}$ with $D_1, D_2 \in S_C \Rightarrow D_1 \not\simeq D_2$ as starting point. The construction of ρ' can be done analogously. □

Proposition 7 (Complete, non-redundant refinement operators II) *Let $\mathcal{L}$ be a description language which contains $\mathcal{AL}$. Let ρ be an arbitrary $\mathcal{L}$ downward refinement operator, where for all concepts C, $\rho^*(C)$ contains only finitely many different concepts equivalent to $\bot$. Then ρ is not complete and non-redundant.*

The restriction mentioned in Proposition 7 is made in order to disallow purely theoretical constructions, as in the proof of Proposition 6, which use syntactic concept extensions that do not alter the semantics of a concept, to ensure non-redundancy. We prove the proposition using a milder, but more technical, restriction.

Proof Let ρ be a complete downward refinement operator, C_{up}, C_{down} be concepts with $C_{\text{down}} \sqsubset C_{\text{up}}$, $\{C \mid C \in \rho^*(C_{\text{up}}), C \equiv C_{\text{down}}\}$ be finite, and S be an infinite set of concepts, which are pairwise incomparable, strictly subsumed by C_{up} and strictly subsuming C_{down}. For instance, we could have $C_{\text{up}} = \top$, $C_{\text{down}} = \bot$ and $S = \{\forall r.A, \forall r.\forall r.A, \dots\}$.

Due to the completeness of ρ, there must be a refinement chain from each concept in S to a concept, which is equivalent to C_{down}. Since there are infinitely many concepts in S, but only finitely many different syntactic representations of C_{down} are reached, there have to be concepts C'_{down}, C_1, and C_2 with $C'_{\text{down}} \equiv C_{\text{down}}$ and $C_1, C_2 \in S$, such that $C'_{\text{down}} \in \rho^*(C_1)$ and $C'_{\text{down}} \in \rho^*(C_2)$.

Because of $C_1 \not\sqsubseteq C_2$ and $C_2 \not\sqsubseteq C_1$, we know $C_1 \notin \rho^*(C_2)$ and $C_2 \notin \rho^*(C_1)$. Hence, there exists a refinement chain from C_{up} to C_{down} through C_1 and a refinement chain from C_{up} to C_{down}, which goes through C_2 and not through C_1. Thus, ρ is redundant. □

Again, a dual result and proof for upward refinement operators can be obtained.

As a consequence of the result, completeness and non-redundancy cannot be combined under reasonable assumptions. Usually, it is desirable to have (weakly) complete operators, but in order to have a full analysis of $\mathcal{L}$ refinement operators we will now also investigate incomplete operators.

Proposition 8 (Incomplete refinement operators) *Let $\mathcal{L}$ be any description language. Then there exists a finite, proper, and non-redundant $\mathcal{L}$ refinement operator.*

Proof The following operator has the desired properties:

$$\rho(C) = \begin{cases} \{\bot\} & \text{if } C \not\equiv \bot \\ \emptyset & \text{otherwise} \end{cases}$$

It is obviously finite, because it maps concepts to sets of cardinality at most 1. It is non-redundant, because it only reaches the bottom concept and there exists no refinement chain of length greater than 2. It is proper, because all concepts, which are not equivalent to the bottom concept strictly subsume the bottom concept.

The corresponding upward operator is:

$$\phi(C) = \begin{cases} \{\top\} & \text{if } C \not\equiv \top \\ \emptyset & \text{otherwise} \end{cases}$$

The arguments for its finiteness, properness, and non-redundancy are analogous to the downward case. □

We can now summarise the results we have obtained so far.

Theorem 1 (Properties of $\mathcal{L}$ refinement operators (I)) *Considering the properties completeness, properness, finiteness, and non-redundancy, the following are maximal sets of properties (in the sense that no other of the mentioned properties can be added) of $\mathcal{L}$ refinement operators ($\mathcal{L} \in \{\mathcal{ALC}, \mathcal{ALCQ}, \mathcal{SHOIN}, \mathcal{SROIQ}\}$):*

1. {*complete*, *finite*}
2. {*complete*, *proper*}
3. {*non-redundant*, *finite*, *proper*}

All results hold under the mild hypothesis stated in Proposition 7.

Proof The theorem is a consequence of the previous results. We have seen that downward and upward operators allow the same combinations of properties, so it is not necessary to distinguish between them. We make a case distinction:

1. The operator is complete. In this case we cannot add non-redundancy (Proposition 7). Finiteness (Proposition 3) and properness (Proposition 5) can be added, but not both (Proposition 4).
2. The operator is not complete. In this case we can add all other properties (Proposition 8). □

A property which we have not yet considered in detail, is weak completeness. Often weak completeness is sufficient, because it allows to search for a good concept starting from $\top$ downwards (top-down approach) or from $\bot$ upwards (bottom-up approach).

We will see that we get different results when considering weak completeness instead of completeness. As a first observation, we see that the arguments in the proof of Proposition 7, which have shown that an $\mathcal{L}$ refinement operator cannot be complete and non-redundant, do no longer apply if we consider weak completeness and non-redundancy.

Proposition 9 (Weakly complete, non-redundant, and proper operators) *Let $\mathcal{L}$ be any description language. Then there exists a weakly complete, non-redundant, and proper $\mathcal{L}$ refinement operator.*

Proof The following operator is weakly complete, non-redundant, and proper:

Let S be a maximal subset of $\{C \mid C \not\equiv \top\}$ with $C_1, C_2 \in S \Rightarrow C_1 \not\sqsupset C_2$.

$$\rho(C) = \begin{cases} S & \text{if } C = \top \\ \emptyset & \text{otherwise} \end{cases}$$

Such a set S as used in the definition of the operator indeed exists. It contains one representative of each equivalence class with respect to weak equality of the set $\{C \mid C \not\equiv \top\}$. The operator is proper, since it contains only mappings of the top concept to concepts, which are not equivalent to top. It is non-redundant, because there is no refinement chain of length greater than 1 and all concepts we reach are pairwise not weakly equal. It is weakly complete, because for every concept, which is not equivalent to $\top$, we can reach an equivalent concept from $\top$ by ρ.

The corresponding upward refinement operator is as follows: Let S be a maximal subset of $\{C \mid C \not\equiv \bot\}$ with $C_1, C_2 \in S \Rightarrow C_1 \not\sqsupset C_2$.

$$\rho(C) = \begin{cases} S & \text{if } C = \bot \\ \emptyset & \text{otherwise} \end{cases}$$ □

The operator just given is obviously not useful in practice, but it suffices for the proof of the proposition.

Proposition 10 (Weakly complete, non-redundant, and finite operators) *Let $\mathcal{L}$ be a description language which allows to express conjunction. Then there exists a weakly complete, non-redundant, and finite $\mathcal{L}$ refinement operator.*

Proof The following operator is weakly complete, non-redundant, and finite:

For an arbitrary concept C, let S_C be a maximal subset of $\{D \mid D \sqsubset \top \text{ and } |D| = \text{number of } \top \text{ occurrences in } C\}$ with $C_1, C_2 \in S_C \implies C_1 \not\equiv C_2$.

$$\rho(C) = \begin{cases} \{\underbrace{\top \sqcap \cdots \sqcap \top}_{n+1 \text{ times } \top}\} \cup S_C & \text{if } C = \underbrace{\top \sqcap \cdots \sqcap \top}_{n \text{ times } \top} \\ \emptyset & \text{otherwise} \end{cases}$$

The operator is finite, because S_C is finite for all concepts C (the number of concepts with a fixed length is finite). It is weakly complete, because every concept C with $C \sqsubset \top$ can be reached from $\top$. This is done by accumulating $\top$ symbols until we have $|C|$ such symbols and then generating C. The operator is furthermore non-redundant, because obviously for all concepts there is exactly one path for reaching the concept via iterated applications of ρ to $\top$. □

The corresponding upward operator is analogous. It works by accumulating $\bot$ symbols instead of $\top$ symbols, and generates concepts which are strictly more general than $\bot$.

Corollary 1 (Weakly complete, proper, and finite operators) *Let $\mathcal{L}$ be any of the description languages $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{SHOIN}$, or $\mathcal{SROIQ}$. Then there exists no weakly complete, finite, and proper $\mathcal{L}$ refinement operator.*

Proof To show this we can use the proof of Proposition 4. There we have shown that in a finite and proper $\mathcal{L}$ refinement operator there exists a concept, which cannot be reached from the $\top$ concept. This means that such an operator cannot be weakly complete. □

The result of the previous observations is that, when requiring only weak completeness instead of completeness, non-redundant operators are possible. The following theorem is the result of the full analysis of the desired properties of $\mathcal{L}$ refinement operators.

Theorem 2 (Properties of refinement operators (II)) *Considering the properties completeness, weak completeness, properness, finiteness, and non-redundancy the following are maximal sets of properties (in the sense that no other of the mentioned properties can be added) of $\mathcal{L}$ refinement operators ($\mathcal{L} \in \{\mathcal{ALC}, \mathcal{ALCQ}, \mathcal{SHOIN}, \mathcal{SROIQ}\}$):*

1. {*weakly complete*, *complete*, *finite*}
2. {*weakly complete*, *complete*, *proper*}
3. {*weakly complete*, *non-redundant*, *finite*}
4. {*weakly complete*, *non-redundant*, *proper*}
5. {*non-redundant*, *finite*, *proper*}

All results hold under the mild hypothesis stated in Proposition 7.

Proof We can do a similar case distinction as in Theorem 1. The first case (complete operator) is analogous except that obviously a complete operator is also weakly complete. For the second case (operator is not complete) we can make a simple case distinction again:

1. The operator is weakly complete. Propositions 9 and 10 have shown that weakly complete operators can be non-redundant and proper as well as non-redundant and finite. Proposition 1 shows that finiteness and properness cannot be combined, so these sets of properties are maximal.
2. The operator is not weakly complete. In this case we can add all remaining properties (Proposition 8), i.e. non-redundancy, finiteness, and properness. □

Note that the restriction of Theorems 1 and 2 to the languages $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{SHOIN}$, and $\mathcal{SROIQ}$ is caused by Proposition 4. The result carries over to other DLs, but might require adapting the proofs. We have provided the formal proofs for these four languages, because they are considered to be fundamental or are the DLs underlying OWL ($\mathcal{SHOIN}$ and $\mathcal{SROIQ}$).

Remark 2 (Subsumption with respect to a TBox) Instead of using subsumption ($\sqsubseteq$) as an ordering over concepts, we can also use subsumption with respect to a TBox $\mathcal{T}$ ($\sqsubseteq_{\mathcal{T}}$). Theorem 2 will also hold in this case. In all negative results, i.e Propositions 4, 7, Corollary 1, we can consider subsumption with respect to an empty TBox as example. In all positive results, i.e. Propositions 3, 5, 8, 9, 10, we can rewrite the example operators by replacing $\sqsubseteq$ with $\sqsubseteq_{\mathcal{T}}$.

Remark 3 (Weak equality) In the definition of redundancy, we used weak equality. Theorem 2 also holds if we consider syntactic equality. Proposition 6 can be simplified. Proposition 7 does not need to be changed. In Propositions 9 and 10 it is straightforward to modify the refinement operators used as positive examples.

5 Designing a refinement operator

We now set out to define a concrete refinement operator using Theorem 2 as starting point, which provides five possible maximal property combinations a refinement operator for description logics can have. We now need to decide which of these five combinations appears to be the one which is most promising in practice. We have to bear in mind that the strongest theoretically possible property combination is not necessarily the most suitable for implementation, as the absence of some theoretical properties is more severe than the absence of others. Indeed, it appears reasonable that it is better not to enforce some properties which would be computationally expensive, if at the same time we can algorithmically limit the negative impact this absence may have. We have a look at each of the properties in turn.

Concerning (weak) completeness, we consider this a very important property to have, since an incomplete operator may fail to converge at all and thus may not return a solution even if one exists. The fifth property combination from Theorem 2 thus appears to be unfavorable.

Concerning finiteness, we will see later in Sect. 6, that having an infinite operator is less critical from a practical perspective, in the sense that this issue can be handled well algorithmically. So it is preferable not to impose finiteness, which allows us to develop a proper operator. This leaves us with the second and fourth property combinations from Theorem 2 to choose from.

As for non-redundancy, this appears to be very difficult to achieve for more complex operators than the one in Proposition 9, which is not useful in practice as it does not structure the search space at all. Consider, for example, the concept $A_1 \sqcap A_2$ which can be reached

from $\top$ via the chain $\top \rightsquigarrow A_1 \rightsquigarrow A_1 \sqcap A_2$, For non-redundancy, the operator would need to make sure that this concept cannot be reached via the chain $\top \rightsquigarrow A_2 \rightsquigarrow A_2 \sqcap A_1$. While there are methods to handle this in such simple cases via normal forms, it becomes more complex for arbitrarily deeply nested structures, where even applying the same replacement leads to redundancy. In our example A_1 is replaced by $A_1 \sqcap A_2$ twice in different order in each chain:

$$\top \rightsquigarrow \forall r_1.A_1 \sqcup \forall r_2.A_1 \rightsquigarrow \forall r_1.A_1 \sqcup \forall r_2.(A_1 \sqcap A_2) \rightsquigarrow \forall r_1.(A_1 \sqcap A_2) \sqcup \forall r_2.(A_1 \sqcap A_2)$$

$$\top \rightsquigarrow \forall r_1.A_1 \sqcup \forall r_2.A_1 \rightsquigarrow \forall r_1.(A_1 \sqcap A_2) \sqcup \forall r_2.A_1 \rightsquigarrow \forall r_1.(A_1 \sqcap A_2) \sqcup \forall r_2.(A_1 \sqcap A_2)$$

To avoid this, an operator would need to regulate when A_1 can be replaced by $A_1 \sqcap A_2$, which appears not to be achievable by syntactic replacement rules. At the same time, we will see in Sect. 6.1 that we can use a computationally inexpensive redundancy check which our experiments have shown to be sufficiently useful in practice.

The arguments just given leave us with the second property combination from Theorem 2. And indeed, reasonable weakly complete operators are often also automatically complete. Consider, for example, the situation where a weakly complete operator ρ allows to refine a concept C to $C \sqcap D$ with some $D \in \rho(\top)$. Then it turns out that this operator is already complete (see proof of Proposition 13). This observation points us again to the second property combination from Theorem 2, which we have already found out to be the most feasible one for developing a refinement operator in our setting.

Summarising, we choose to develop a weakly complete, complete, and proper refinement operator, and will show that we can handle issues arising from infinity and redundancy well algorithmically.

We proceed as follows: First, we define a refinement operator and prove its completeness. We then extend it to a complete and proper operator. Section 6 will show how we handle the problems of redundancy and infinity in the learning algorithm.

5.1 Definition of the operator

We now define the operator and prove that it is indeed a downward refinement operator. For each $A \in N_C$, we define:

$$sh_{\downarrow}(A) = \{A' \in N_C \mid A' \sqsubset A, \text{ there is no } A'' \in N_C \text{ with } A' \sqsubset_{\mathcal{T}} A'' \sqsubset_{\mathcal{T}} A\}$$

$sh_{\downarrow}(\top)$ is defined analogously for $\top$ instead of A. $sh_{\uparrow}(A)$ is defined analogously for going upward in the subsumption hierarchy. (sh stands for subsumption hierarchy.)

We do the same for roles, i.e.:

$$sh_{\downarrow}(r) = \{r' \mid r' \in N_R, r' \sqsubset r, \text{ there is no } r'' \in N_R \text{ with } r' \sqsubset_{\mathcal{T}} r'' \sqsubset_{\mathcal{T}} r\}$$

$domain(r)$ denotes the domain of a role r and $range(r)$ the range of a role r. A range axiom links a role to a concept. It asserts that the role fillers must be instances of a given concept. Analogously, domain axioms restrict the first argument of role assertions to a concept.

We define:

$$ad(r) = \text{an } A \text{ with } A \in \{\top\} \cup N_C \text{ and } domain(r) \sqsubseteq A \text{ and there does not exist an } A' \text{ with } domain(r) \sqsubseteq A' \sqsubset A$$

$ar(r)$ is defined analogously using *range* instead of *domain*. *ad* stands for atomic domain and *ar* stands for atomic range. We assign exactly one atomic concept as domain/range of a role. Since using atomic concepts as domain and range is very common, *domain* and *ad* as well as *range* and *ar* will usually coincide.

The set app_B of applicable properties with respect to an atomic concept B is defined as:

$$app_B = \{r | r \in N_R, ad(r) = A, A \sqcap B \not\equiv \bot\}$$

To give an example, for the concept `Person`, we have that the role `hasChild` with $ad($`hasChild`$) =$ `Person` is applicable, whereas the role `hasAtom` with $ad($`hasAtom`$) =$ `ChemicalCompound` is not applicable (assuming `Person` and `ChemicalCompound` are disjoint). We will use this to restrict the search space by ruling out unsatisfiable concepts.

The set of most general applicable roles mgr_B with respect to a concept B is defined as:

$$mgr_B = \{r \mid r \in app_B, \text{ there is no } r' \text{ with } r \sqsubset r', r' \in app_B\}$$

M_B with $B \in \{\top\} \cup N_C$ is defined as the union of the following sets:

- $\{A \mid A \in N_C, A \sqcap B \not\equiv \bot, A \sqcap B \not\equiv B, \text{ there is no } A' \in N_C \text{ with } A \sqsubset A'\}$
- $\{\neg A \mid A \in N_C, \neg A \sqcap B \not\equiv \bot, \neg A \sqcap B \not\equiv B, \text{ there is no } A' \in N_C \text{ with } A' \sqsubset A\}$
- $\{\exists r.\top \mid r \in mgr_B\}$
- $\{\forall r.\top \mid r \in mgr_B\}$

M_B is the set of refinements of the $\top$ concept excluding the use of disjunction ($\sqcup$) and restricted to those which are not disjoint with B.

The operator ρ is defined in Fig. 1. Note that ρ delegates to an operator ρ_B with $B = \top$ initially, where B is set to the atomic range of roles contained in the input concept when the operator recursively traverses the structure of the concept. The index B in the operator (and the set M above) is used to rule out concepts which are disjoint with B.

We use the following notions for different kinds of refinement steps in ρ:

1. $\overset{\sqcap}{\rightsquigarrow}$: add an element conjunctively (cases 3, 4, 5, 6, 8 in the definition of ρ_B in Fig. 1)
2. $\overset{\top}{\rightsquigarrow}$: refine the top concept (case 2 in the definition of ρ_B in Fig. 1)
3. $\overset{A}{\rightsquigarrow}$: refine an atomic concept (case 3 in the definition of ρ_B in Fig. 1)
4. $\overset{\neg A}{\rightsquigarrow}$: refine a negated atomic concept (case 4 in the definition of ρ_B in Fig. 1)
5. $\overset{r}{\rightsquigarrow}$: refine a role (cases 5, 6 in the definition of ρ_B in Fig. 1)

If a concept is refined to some other concept using ρ, exactly one of these five steps is performed. We assume that conjunctions are never nested in conjunctions and disjunctions are never nested in disjunctions, e.g. $A_1 \sqcap (A_2 \sqcap \exists r.\top)$ is written as $A_1 \sqcap A_2 \sqcap \exists r.\top$ instead.

Proposition 11 *ρ is an $\mathcal{ALC}$ downward refinement operator.*

Proof We have to show that $D \in \rho(C)$ implies $D \sqsubset_{\mathcal{T}} C$. We show $D \sqsubset C$, i.e. consider an empty TBox. Since $D \sqsubset C$ implies $D \sqsubset_{\mathcal{T}} C$ by the definition of subsumption, the desired result follows.

We prove by structural induction of $\mathcal{ALC}$ concepts in negation normal form. Obviously, in all cases where $D = C \sqcap C'$, i.e. C is extended conjunctively by a concept C' we have $D \sqsubset C$, so these cases are ignored.

- $C = \bot$: $D \in \rho(C)$ is impossible, because $\rho(\bot) = \emptyset$.

$$\rho(C) = \begin{cases} \{\bot\} \cup \rho_\top(C) & \text{if } C = \top \\ \rho_\top(C) & \text{otherwise} \end{cases}$$

$$\rho_B(C) = \begin{cases} \emptyset & \text{if } C = \bot \\ \{C_1 \sqcup \cdots \sqcup C_n \mid C_i \in M_B\ (1 \le i \le n)\} & \text{if } C = \top \\ \{A' \mid A' \in sh_\downarrow(A)\} & \text{if } C = A\ (A \in N_C) \\ \quad \cup \{A \sqcap D \mid D \in \rho_B(\top)\} & \\ \{\neg A' \mid A' \in sh_\uparrow(A)\} & \text{if } C = \neg A\ (A \in N_C) \\ \quad \cup \{\neg A \sqcap D \mid D \in \rho_B(\top)\} & \\ \{\exists r.E \mid A = ar(r), E \in \rho_A(D)\} & \text{if } C = \exists r.D \\ \quad \cup \{\exists r.D \sqcap E \mid E \in \rho_B(\top)\} & \\ \quad \cup \{\exists s.D \mid s \in sh_\downarrow(r)\} & \\ \{\forall r.E \mid A = ar(r), E \in \rho_A(D)\} & \text{if } C = \forall r.D \\ \quad \cup \{\forall r.D \sqcap E \mid E \in \rho_B(\top)\} & \\ \quad \cup \{\forall r.\bot \mid & \\ \qquad D = A \in N_C \text{ and } sh_\downarrow(A) = \emptyset\} & \\ \quad \cup \{\forall s.D \mid s \in sh_\downarrow(r)\} & \\ \{C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \cdots \sqcap C_n \mid & \text{if } C = C_1 \sqcap \cdots \sqcap C_n \\ \qquad D \in \rho_B(C_i), 1 \le i \le n\} & (n \ge 2) \\ \{C_1 \sqcup \cdots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \cdots \sqcup C_n \mid & \text{if } C = C_1 \sqcup \cdots \sqcup C_n \\ \qquad D \in \rho_B(C_i), 1 \le i \le n\} & (n \ge 2) \\ \quad \cup \{(C_1 \sqcup \cdots \sqcup C_n) \sqcap D \mid & \\ \qquad D \in \rho_B(\top)\} & \end{cases}$$

Fig. 1 Definition of the refinement operator ρ

- $C = \top$: $D \sqsubseteq C$ is trivially true.
- $C = A$ $(A \in N_C)$: $D \in \rho(C)$ implies that D is also an atomic concept and $D \sqsubset_\mathcal{T} C$. Thus $D \sqsubseteq C$.
- $C = \neg A$: $D \in \rho(C)$ implies that D is of the form $\neg A'$ with $A \sqsubset_\mathcal{T} A'$. $A \sqsubset_\mathcal{T} A'$ implies $\neg A' \sqsubset_\mathcal{T} \neg A$ by the semantics of negation. Thus, $D \sqsubseteq C$.
- $C = \exists r.C'$: $D \in \rho(C)$ implies that D is of the form $\exists r.D'$ or $\exists s.C'$. For the former case, we have $D' \sqsubseteq C'$ by induction. (For existential restrictions $\exists r.E \sqsubseteq \exists r.E'$ if $E \sqsubset E'$ holds in general.) For the latter case, we obviously have $\exists s.C' \sqsubseteq \exists r.C'$, because $s \sqsubset r$.
- $C = \forall r.C'$: This case is analogous to the previous one. Here we use the results $\forall r.E \sqsubseteq \forall r.E'$ $(E \sqsubset_\mathcal{T} E')$ and $\forall s.C' \sqsubseteq \forall r.C'$ $(s \sqsubset r)$.
- $C = C_1 \sqcap \cdots \sqcap C_n$: In this case one element of the conjunction is refined, so $D \sqsubseteq C$ follows by induction.
- $C = C_1 \sqcup \cdots \sqcup C_n$: In this case one element of the disjunction is refined, so $D \sqsubseteq C$ follows by induction. □

A distinguishing feature of ρ compared to other refinement operators (Badea and Nienhuys-Cheng 2000; Esposito et al. 2004) for learning concepts in DLs is that it makes use of the subsumption and role hierarchy, e.g. for concepts $A_2 \sqsubset A_1$, we reach A_2 via $\top \rightsquigarrow A_1 \rightsquigarrow A_2$. This way, we can stop the search if A_1 is already too weak and, thus, make

better use of TBox knowledge. The operator also uses domain and range of roles to reduce the search space. This is similar to mode declarations in Aleph, Progol, and other ILP programs. However, in DL knowledge bases and OWL ontologies, domain and range are usually explicitly given, so there is no need to define them manually. Overall, the operator supports more structures than those in Badea and Nienhuys-Cheng (2000), Esposito et al. (2004) and tries to intelligently incorporate background knowledge. Section 5.4 describes further extensions of the operator, which increase its expressivity such that it can handle most OWL constructs.

Note that ρ is infinite. The reason is that the set M_B is infinite and, furthermore, we put no bound on the number of elements in the disjunctions, which are refinements of the top concept. Another point is that the operator requires reasoner requests for calculating M_B. However, the number of requests is fixed, so—assuming the results of those requests are cached—the reasoner is only needed in an initial phase, i.e. during the first calls to the refinement operator. This means that, apart from this initial phase, the refinement operator performs only syntactic rewriting rules.

5.2 Completeness of the operator

To investigate the completeness of the operator, we define a set $S_\downarrow$ of $\mathcal{ALC}$ concepts in negation normal form as follows:

Definition 12 ($S_\downarrow$) We define $S_\downarrow = S'_\downarrow \cup \{\bot\}$, where $S'_\downarrow$ is defined as follows:

1. If $A \in N_C$ then $A \in S'_\downarrow$ and $\neg A \in S'_\downarrow$.
2. If $r \in N_R$ then $\forall r.\bot \in S'_\downarrow$, $\forall r.\top \in S'_\downarrow$, $\exists r.\top \in S'_\downarrow$.
3. If $C, C_1, \ldots, C_m$ are in $S'_\downarrow$ then the following concepts are also in $S'_\downarrow$:
   - $\exists r.C$ if $C \sqcap ar(r) \not\equiv \bot$
   - $\forall r.C$ if $C \sqcap ar(r) \not\equiv \bot$
   - $C_1 \sqcap \cdots \sqcap C_m$
   - $C_1 \sqcup \cdots \sqcup C_m$ if for all i $(1 \leq i \leq m)$ C_i is not of the form $D_1 \sqcap \cdots \sqcap D_n$ where all D_j $(1 \leq j \leq n)$ are of the form $E_1 \sqcup \cdots \sqcup E_p$.

In $S_\downarrow$, we do not use the $\top$ and $\bot$ symbols directly and we make a restriction on disjunctions, i.e. we do not allow that elements of a disjunction are conjunctions, which in turn only consist of disjunctions. It can be shown that for all $\mathcal{ALC}$ concepts there exists an equivalent concept in $S_\downarrow$.

Lemma 3 ($S_\downarrow$) *For all $\mathcal{ALC}$ concepts C there exists a concept $D \in S_\downarrow$ such that $D \equiv C$.*

Proof We assume C is in negation normal form. The restriction $C \sqcap ar(r) \not\equiv \bot$ for concepts of the form $\exists r.C$ and $\forall r.C$ does not exclude any relevant concepts, because we have $\exists r.C \equiv \exists r.(C \sqcap ar(r))$ and $\forall r.C \equiv \forall r.(C \sqcap ar(r))$ in general. Replacing $C \sqcap ar(r)$ by $\bot$ yields $\exists r.\bot \equiv \bot$ and $\forall r.\bot$. Both, $\bot$ and $\forall r.\bot$, are already in $S_\downarrow$.

The proof consists of three steps: First, we will eliminate $\top$ symbols unless they occur in existential restrictions (because in Definition 12 $\exists r.\top$ is used in the induction base opposed to using $\top$ directly). After that, we do something similar with the bottom symbol. In a third step we will eliminate disjunctions violating the criterion in Definition 12. After these three steps, we obtain a concept, which is in $S_\downarrow$.

We eliminate $\top$-symbols by applying the following rewrite rules:

$$C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap \top \sqcap C_{i+1} \sqcap \cdots \sqcap C_n \rightarrow C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap C_{i+1} \sqcap \cdots \sqcap C_n$$
$$C_1 \sqcup \cdots \sqcup C_{i-1} \sqcup \top \sqcup C_{i+1} \sqcup \cdots \sqcup C_n \rightarrow \top$$
$$\forall r.\top \rightarrow \top$$

Obviously, these $\top$-elimination steps preserve equivalence. We exhaustively apply these steps (since every step reduces the length of the concept there can be only finitely many such steps) to get a concept C'. Note that $C' \neq \top$ (otherwise $C' \equiv C \equiv \top$) and in C' the top concept only appears in existential restrictions, i.e. in the form $\exists r.\top$.

$\bot$ symbols are eliminated by the following rewrite rules:

$$C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap \bot \sqcap C_{i+1} \sqcap \cdots \sqcap C_n \rightarrow \bot$$
$$C_1 \sqcup \cdots \sqcup C_{i-1} \sqcup \bot \sqcup C_{i+1} \sqcup \cdots \sqcup C_n \rightarrow C_1 \sqcup \cdots \sqcup C_{i-1} \sqcup C_{i+1} \sqcup \cdots \sqcup C_n$$
$$\exists r.\bot \rightarrow \bot$$

These steps also preserve equivalence. After exhaustively applying these steps we either get the $\bot$ symbol itself (which is in $S_\downarrow$) or the $\bot$ symbol only appears in universal restrictions, i.e. in the form $\forall r.\bot$.

Next we have to eliminate disjunctions, which do not satisfy Definition 12. Say we have such a disjunction $C_1 \sqcup \cdots \sqcup C_m$. Then there is a C_i $(1 \leq i \leq m)$, which is a conjunction consisting only of disjunctions. Without loss of generality we assume $i = 1$ (the order of elements in a disjunction is not important), i.e. we can write $C_1 = D_1 \sqcap \cdots \sqcap D_n$ and $D_1 = E_1 \sqcup \cdots \sqcup E_p$. This means we can apply the following equivalence preserving rewriting rule:

$$((E_1 \sqcup \cdots \sqcup E_p) \sqcap D_2 \sqcap \cdots \sqcap D_n) \sqcup C_2 \sqcup \cdots \sqcup C_m$$
$$\rightarrow (E_1 \sqcap D_2 \sqcap \cdots \sqcap D_n) \sqcup \cdots \sqcup (E_p \sqcap D_2 \sqcap \cdots \sqcap D_n) \sqcup C_2 \cdots \sqcup C_m$$

Note that E_i $(1 \leq i \leq p)$ cannot be a disjunction. Let C_1' be the replacement of C_1 after applying the rewriting rule. Obviously, C_1' is no more a disjunction where an element is a conjunction of disjunctions (because all E_i are not disjunctions). If we apply this rule to all applicable C_i $(1 \leq i \leq m)$, then we obtain a concept C'' equivalent to $C_1 \sqcup \cdots \sqcup C_m$, which is in $S_\downarrow$.

Hence, we have shown that we can construct a concept $C'' \equiv C' \equiv C$ with $C'' \in S_\downarrow$, which completes the proof. □

Lemma 4 *For all concepts B with $B \in \{\top\} \cup N_C$, all $\mathcal{ALC}$ concepts $C \in S'_\downarrow$ with $C \sqsubset B$, which satisfy the following restriction, can be reached from $\top$ by ρ_B:*

- *C is of the form $C_1 \sqcup \cdots \sqcup C_m$ and for all i $(1 \leq i \leq m)$ $C_i \sqcap B \not\equiv \bot$*
- *C is not of the form $C_1 \sqcup \cdots \sqcup C_m$ and $C \sqcap B \not\equiv \bot$*

Proof We prove by induction over the structure of concepts in $S'_\downarrow$. Parts of the rather involved proof are only sketched.

- Induction Base: An atomic concept A can be reached from $\top$ by a refinement chain of the following form:

$$\top \overset{\top}{\rightsquigarrow} A_1 \overset{A}{\rightsquigarrow} \cdots \overset{A}{\rightsquigarrow} A_n \overset{A}{\rightsquigarrow} A'$$

The operator descends the subsumption hierarchy. Since A' is more special than A, we can always reach it (unless A and A' are disjoint, which is excluded by the induction hypothesis). If B is an atomic concept, then the definition of M_B ensures that A_1 is a concept, which is subsumed by B. Since there are only finitely many atomic concepts, A will be reached in a finite number of steps. Negated atomic concepts can be handled analogously.

$\forall r.\bot$ can be reached by descending the subsumption and role hierarchy:

$$\top \overset{\top}{\rightsquigarrow} \forall s.\top \overset{r}{\rightsquigarrow} \cdots \overset{r}{\rightsquigarrow} \forall r.\top \rightsquigarrow \forall r.A_1 \rightsquigarrow \cdots \rightsquigarrow \forall r.A_n \rightsquigarrow \forall r.\bot$$

To reach $\exists r.\top$, the following chain can be used:

$$\top \overset{\top}{\rightsquigarrow} \exists s.\top \overset{r}{\rightsquigarrow} \cdots \overset{r}{\rightsquigarrow} \exists r.\top$$

$\forall r.\top$ can be handled analogously.

– Induction Step:
  – $\exists r.C$: We have $\top \overset{\top}{\rightsquigarrow} \exists s.\top \overset{r}{\rightsquigarrow} \cdots \overset{r}{\rightsquigarrow} \exists r.\top$ and by induction we can reach C with $C \sqcap B \not\equiv \bot$ from $\top$ by ρ_A where $A = ar(r)$.
  – $\forall r.C$: Analogously to $\exists r.C$.
  – $C_1 \sqcap \cdots \sqcap C_m$: We know $C_1 \sqcap \cdots \sqcap C_m \sqcap A \not\equiv \bot$, which implies $C_i \sqcap A \not\equiv \bot$ for all i $(1 \leq i \leq m)$. This means we know that C_1 can be reached from $\top$ using ρ_B by induction. Thus, we can first refine to C_1 and then add all other concepts to the conjunction stepwise:

$$\top \rightsquigarrow^* C_1 \rightsquigarrow^* C_1 \sqcap C_2 \rightsquigarrow^* C_1 \sqcap \cdots \sqcap C_m$$

  Note that ρ does not allow to extend a conjunction directly, but instead in all other concept structures the operator allows to append an element conjunctively, e.g. $C_1 \sqcap C_2$ can be refined to $C_1 \sqcap (C_2 \sqcap D) \equiv C_1 \sqcap C_2 \sqcap D$ with $D \in \rho_B(\top)$.
  – $C_1 \sqcup \cdots \sqcup C_m$: We have to show $C_i \in \rho^*(m)$ for an $m \in M_B$. We do this by a case distinction on the structure of C_i for an arbitrary i $(1 \leq i \leq m)$.

    If C_i is an atomic concept A: In this case, we pick an atomic concept A_1 (see point 1 in the definition of M_B on page 223) and refine it:

$$A_1 \overset{A}{\rightsquigarrow} \cdots \overset{A}{\rightsquigarrow} A_n \overset{A}{\rightsquigarrow} A$$

    This works exactly as in the case $C = A$ above.

    Similarly, if C_i is of the form $\neg A$, we pick a concept $\neg A_2$ according to point 2 in the definition of M_B and refine to $\neg A$. The cases $\exists r.\top$, $\forall r.\top$, $\forall r.\bot$, $\exists r.C$, $\forall r.C$ can be handled analogously.

    If C_i is of the form $D_1 \sqcap \cdots \sqcap D_n$, then we know that according to the definition of $S'_{\downarrow}$ there is a D_j which is not a disjunction (and obviously not a conjunction because C_i is a conjunction). This means that D_j can be handled by any of the previous cases. Having refined to D_j, we can then add all other D_k $(1 \leq k \leq n, k \neq j)$ to the conjunction.

    Summed up, we can refine to C by picking the right concept in M_B for each element of the disjunction and then refining further:

$$\begin{aligned} \top \ &\rightsquigarrow \ E_1 \sqcup \cdots \sqcup E_m \quad (E_l \in M_B, 1 \leq l \leq m) \\ &\rightsquigarrow^* C_1 \sqcup E_2 \sqcup \cdots \sqcup E_m \rightsquigarrow C_1 \sqcup C_2 \sqcup E_3 \sqcup \cdots \sqcup E_m \rightsquigarrow \cdots \rightsquigarrow C_1 \sqcup \cdots \sqcup C_m \end{aligned}$$

□

This allows us to show weak completeness by proving that every element in $S_\downarrow$ can be reached from $\top$ by ρ.

Proposition 12 (Weak completeness of ρ) *ρ is weakly complete.*

Proof We have to show that for all concepts C with $C \sqsubset_\mathcal{T} \top$ a concept E with $E \equiv C$ can be reached from $\top$ by ρ. Due to Lemma 3, it is sufficient to show that all concepts in $S_\downarrow$ can be reached from $\top$ by ρ.

Lemma 4 proves that we can reach all concepts in $S'_\downarrow$ from $\top$ using $\rho_\top$, because using $B = \top$ the restriction made in this lemma, is always satisfied:

- If C is not a disjunction: $C \sqcap \top \not\equiv \bot$ is always true unless $C \equiv \bot$, but $\bot$ can be reached from $\top$ using ρ (see below).
- If C is a disjunction: $C_i \sqcap \top \not\equiv \bot$ is always true unless $C_i \equiv \bot$, but in this case C is not in $S_\downarrow$ ($\bot$ cannot be an element of a disjunction in $S_\downarrow$).

The only element in $S_\downarrow$, which is not in $S'_\downarrow$ is $\bot$, which can be reached in one refinement step from $\top$ using ρ. Therefore, all concepts in $S_\downarrow$ can be reached from $\top$ using ρ. □

Using this, we can prove completeness.

Proposition 13 (Completeness of ρ) *ρ is complete.*

Proof Let C and D be arbitrary $\mathcal{ALC}$ concepts in $S_\downarrow$ with $C \sqsubset_\mathcal{T} D$. To prove completeness of ρ, we have to show that there exists a concept E with $E \equiv C$ and $E \in \rho^*(D)$. $E = D \sqcap C$ satisfies this property. We obviously have $E = D \sqcap C \equiv C$, because of $C \sqsubset_\mathcal{T} D$. We know that ρ allows to extend concepts conjunctively by refinements of the top concept. Hence, we know that $D \sqcap C$ can be reached from D for all concepts C by the weak completeness result for ρ. Thus, ρ is complete. □

5.3 Achieving properness

The operator ρ is not proper, for instance it allows the following refinements:

$$\top \rightsquigarrow \exists r.\top \sqcup \forall r.\top \rightsquigarrow \exists r.\top \sqcup \forall r.A_1 \rightsquigarrow \exists r.A_2 \sqcup \forall r.A_1$$

In this chain, the first three concepts are equivalent. One could try to modify ρ, such that it becomes proper. However, the last refinement can be reached only via improper refinement steps. This means that modifying the operator is likely to lead to incompleteness as e.g. $\exists r.A_2 \sqcup \forall r.A_1$ would need to be reached from $\top$ (Proposition 5 shows that operators can be complete and proper, but such operators do not structure the search space well). Indeed, there is no tractable structural subsumption algorithm for $\mathcal{ALC}$ (Baader et al. 2007), which indicates that it is hard to define a proper operator just by syntactic rewriting rules.

So, instead of modifying ρ directly, we allow it to be improper, but consider the closure $\rho^{cl}_\downarrow$ of ρ (Badea and Nienhuys-Cheng 2000).

Definition 13 ($\rho^{cl}_\downarrow$) $\rho^{cl}_\downarrow$ is defined as follows: $D \in \rho^{cl}_\downarrow(C)$ iff there exists a refinement chain

$$C \rightsquigarrow_\rho C_1 \rightsquigarrow_\rho \cdots \rightsquigarrow_\rho C_n = D$$

such that $C \not\equiv D$ and $C_i \equiv C$ for $i \in \{1, \ldots, n-1\}$.

$\rho^{cl}_{\downarrow}$ is proper by definition. It also inherits the completeness of ρ, since we do not disallow any refinement steps, but only check whether they are improper. We already know that ρ is infinite, so it is clear that we cannot consider all refinements of a concept at a time. Therefore, in practice we will always compute all refinements of a concept up to a given length. A flexible algorithm will allow this length limit to be increased if necessary. Using this technique, an infinite operator can be handled. However, we have to make sure that all refinements of $\rho^{cl}_{\downarrow}$ up to a given length are computable in finite time. To show this, we need the following lemma.

Lemma 5 (ρ does not reduce length) *$D \in \rho(C)$ implies $|D| \geq |C|$. Furthermore, there are no infinite refinement chains of the form $C_1 \rightsquigarrow_\rho C_2 \rightsquigarrow_\rho \cdots$ with $|C_1| = |C_2| = \cdots$, i.e. after a finite number of steps we reach a strictly longer concept.*

Proof To show the first statement we need to observe the steps, which are performed by ρ. As mentioned before, ρ can do one of five things in each refinement step:

1. add an element conjunctively ($\overset{\sqcap}{\rightsquigarrow}$)
2. refine the top concept ($\overset{\top}{\rightsquigarrow}$) not including refinements of the form $\top \rightsquigarrow A$
3. refine an atomic concept ($\overset{A}{\rightsquigarrow}$) including refinements of the form $\top \rightsquigarrow A$
4. refine a negated atomic concept ($\overset{\neg A}{\rightsquigarrow}$)
5. refine a role ($\overset{r}{\rightsquigarrow}$)

Steps 1 and 2 result in a concept with greater length (for this reason we excluded $\top \rightsquigarrow A$ from step 2). Step 3 to 5 result in a concept with the same length. This proves the claim made in the first sentence.

The second claim follows from the fact that there is just a finite number of atomic concepts and roles (N_C, N_R are finite) and there are only finitely many occurrences of an atomic concept within any concept. Hence, there are no infinite refinement chains using only steps 3 to 5. Thus, after a finite number of refinements, step 1 or 2 is used, which produces a longer concept. □

Proposition 14 *For all concepts C in negation normal form and all natural numbers n, the set $\{D \mid D \in \rho^{cl}_{\downarrow}(C), |D| \leq n\}$ can be computed in finite time.*

Proof Due to Lemma 5, we know that for all concepts D in the set, there exists an m such that $|D| > |C|$ with $D \in \rho^m(C)$. Obviously, a concept has only finitely many refinements up to a fixed length. If we consider all refinement chains of a concept C by ρ up to length n as a tree, then this tree is finite (there are only finitely many concepts of length $\leq n$ and any such concept can be reached by a finite refinement chain). The set $\{D \mid D \in \rho^{cl}_{\downarrow}(C), |D| \leq n\}$ is a subset of the nodes of this tree. Hence, it can be computed in finite time. □

Due to Proposition 14 we can use $\rho^{cl}_{\downarrow}$ in a learning algorithm. For computing $\rho^{cl}_{\downarrow}$ up to n, it is sufficient to apply the operator until a non-equivalent concept is reached. By a straightforward analysis of the refinement steps, one can show that in the worst case after $O(|N_C| \cdot |N_R| \cdot |C|)$ steps a refinement of greater length will be reached, which bounds the complexity of computing the closure.

5.4 Cardinality restrictions and concrete role support

This section describes an extension of the presented refinement operator with cardinality restrictions and support for boolean and double concrete roles (called data properties in OWL). Syntax and semantics of those constructs can be found in Table 2. They can be used to construct concepts such as $\texttt{Person} \sqcap \texttt{height} \geq 1.85$ (persons taller than 1.85), $\texttt{Student} \sqcap \geq 3\ \texttt{hasCar}.\top$ (students with more than 3 cars), $\texttt{Patient} \sqcap \texttt{pregnancyTest} = \mathit{true}$ (patients with positive pregnancy test). They are described as extensions here, because we wanted to keep the presentation of the refinement operator brief. Furthermore, the operator becomes incomplete if those extensions are included. For instance, when enabling double concrete role support, Person can be refined to $\texttt{Person} \sqcap \texttt{height} \geq x$, where x is one of finitely many values determined by analysing the knowledge base (described below). Since the set of real numbers is infinite, this means we cannot—and of course do not want to—reach all concepts of the form $\texttt{Person} \sqcap \texttt{height} \geq x$ and, thus, the operator is not complete.

To support the constructs listed in Table 2, the refinement operator was extended as follows.

Analogously to the already introduced sets N_C, N_R, and N_I in Sect. 2, we introduce the following notions: The set N_{CR} stands for the set of all concrete roles, the set N_{BCR} stands for the set of all boolean concrete roles, and the set N_{DCR} stands for the set of all double concrete roles.

The sets *mgb* for boolean concrete roles and *mgd* for double concrete roles are defined analogously to *mgr* (see Sect. 5.1).

Table 2 Overview of the syntax and semantics of concrete role constructs supported by the refinement operator. See Table 1 for $\mathcal{ALC}$ syntax and semantics. $card\{\dots\}$ denotes set cardinality

Syntax	Construct
r	abstract role
b	boolean concrete role
d	double concrete role
$\leq n\ r.C$	max. cardinality restriction
$\geq n\ r.C$	min. cardinality restriction
$b = \mathit{true}$	exists boolean true value restriction
$b = \mathit{false}$	exists boolean false value restriction
$d \leq v$	exists double max. restriction ($v \in \mathcal{R}$)
$d \geq v$	exists double min. restriction ($v \in \mathcal{R}$)
Syntax	**Semantics**
r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
b	$b^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \{\mathit{false}, \mathit{true}\}$
d	$d^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \mathcal{R}$
$\leq n\ r.C$	$(\leq n\ r.C)^{\mathcal{I}} = \{a \mid card\{b \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\} \leq n\}$
$\geq n\ r.C$	$(\geq n\ r.C)^{\mathcal{I}} = \{a \mid card\{b \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\} \geq n\}$
$b = \mathit{true}$	$(b = \mathit{true})^{\mathcal{I}} = \{a \mid (a, \mathit{true}) \in b^{\mathcal{I}}\}$
$b = \mathit{false}$	$(b = \mathit{false})^{\mathcal{I}} = \{a \mid (a, \mathit{true}) \in b^{\mathcal{I}}\}$
$d \leq v$	$(d \leq v)^{\mathcal{I}} = \{a \mid \exists v'.(a,v') \in d^{\mathcal{I}} \text{ and } v' \leq v\}$
$d \geq v$	$(d \geq v)^{\mathcal{I}} = \{a \mid \exists v'.(a,v') \in d^{\mathcal{I}} \text{ and } v' \geq v\}$

Let $values_d$ ($d \in N_{DCR}$) be a list containing the following double numbers in ascending order: $\{t \mid \mathcal{K} \models d(a,t)\}$. $values_d[i]$ denotes the i-th element in this list. $\#splits_d \in \mathcal{N}$ is a user defined parameter of the operator for specifying how many refinement steps should be used to traverse the values of double concrete roles. The list $splits_d$ contains the following double numbers in ascending order:

$$\left\{ t_j \mid i = \frac{\#values_d}{\#splits_d + 1}, t = \frac{1}{2}(values_d[\lfloor i \cdot j \rfloor] + values_d[\lfloor i \cdot j \rfloor + 1]) \text{ for } 1 \leq j \leq \#splits_d \right\}$$

Again, we use $splits_d[i]$ to refer to the i-th element in this list. Clearly, one can employ different strategies for finding sensible splitting values.

$\mathrm{mf}_r = \max_{a \in N_I} |\{b \mid \mathcal{K} \models r(a,b)\}|$ is the maximum number of role fillers of a role r. We use this as upper limit for cardinality restrictions.

The set M_B is extended by the following sets:

- $\{\leq \mathrm{mf}_r\ r.\top \mid r \in mgr_B\}$
- $\{b = true \mid b \in mgb_B\}$
- $\{b = false \mid b \in mgb_B\}$
- $\{d \geq v \mid d \in mgd_B, v = splits_d[\#splits_d]\}$
- $\{d \leq v \mid d \in mgd_B, v = splits_d[1]\}$

Finally, the refinement operator is extended as follows for handling the novel constructs:

$$\rho_B(C) = \begin{cases} \geq 2\ r.D & \text{if } C = \exists r.D \\ \{\geq n+1\ r.D \mid n < \mathrm{mf}_r\} & \text{if } C = \geq n\ r.D \\ \quad \cup \{\geq n\ r.E \mid E \in \rho_B(D)\} & \\ \{\leq n-1\ r.D \mid n > 1\} & \text{if } C = \leq n\ r.D \\ \quad \cup \{\leq n\ r.E \mid E \in \rho_B(D)\} & \\ \emptyset & \text{if } C = (b = true) \\ \emptyset & \text{if } C = (b = false) \\ \{d \geq w \mid v = splits_d[i], i > 1, & \text{if } C = (d \geq v) \\ \qquad w = splits_d[i-1]\} & \\ \{d \leq w \mid v = splits_d[i], i < \#splits_d, & \text{if } C = (d \leq v) \\ \qquad w = splits_d[i+1]\} & \end{cases}$$

We used the presented extensions in the carcinogenesis benchmark (Sect. 7.2). They extend the expressiveness of the target language close to OWL.

6 The learning algorithm

So far, we have designed a complete and proper operator. Unfortunately, such an operator has to be redundant and infinite by Theorem 1. We will now describe how to deal with these problems and define the overall learning algorithm.

6.1 Redundancy elimination

A learning algorithm can be constructed as a combination of a refinement operator, which defines how the search tree can be built, and a search algorithm, which controls how the tree

is traversed. The search algorithm specifies which nodes have to be expanded. Whenever the search algorithm encounters a node in the search tree, it could check whether a weakly equal concept already exists in the search tree. If yes, then this node is ignored, i.e. it will not be expanded further and it will not be evaluated. This removes all redundancies, since every concept exists at most once in the search tree.[8] We can still reach all concepts, because we have $\rho^{cl}_{\downarrow}(C) \simeq \rho^{cl}_{\downarrow}(D)$ if $C \simeq D$, i.e. $\rho^{cl}_{\downarrow}$ handles weakly equal concepts in the same way. However, this redundancy elimination approach is computationally expensive if performed naively. Hence, we considered it worthwhile to investigate how it can be handled efficiently.

Note, that we consider weak equality instead of equality here, e.g. we have $A_1 \sqcap A_2 \neq A_2 \sqcap A_1$, but $A_1 \sqcap A_2 \simeq A_2 \sqcap A_1$. We do this, because having $A_1 \sqcap A_2$ and $A_2 \sqcap A_1$—while not being syntactically equal—can still be considered redundant and should be avoided. In conjunctions and disjunctions, this raises the problem that we have to guess which pairs of elements are equal to determine whether two concepts are weakly equal. One way to solve this problem is to define an ordering over concepts and require the elements of disjunctions and conjunctions to be ordered accordingly. This eliminates the guessing step and allows to check weak equality in linear time. There are different ways to define a linear order $\preceq$ over concepts, and we have shown that it is also possible to do it in such a way that deciding $\preceq$ for two concepts is polynomial and transforming a concept in negation normal form to $\preceq$ *ordered negation normal form*, i.e. elements in conjunctions and disjunctions are ordered with respect to $\preceq$, can be done in polynomial time—for brevity we omit the details. It is thus reasonable to assume that every concept occurring in our search tree can be transformed to ordered negation normal form with respect to some linear order over concepts. We can then maintain an ordered set of concepts occurring in the search tree. Checking weak equality of a concept C with respect to a search tree containing n concepts will then only require $\log n$ comparisons (binary search), where each comparison needs only linear time. Taking into account the complexity of instance checks (EXPTIME for $\mathcal{ALC}$, NEXPTIME for $\mathcal{SHOIN}(D)$ and OWL-DL), which we can avoid (compared to an algorithm without redundancy check), redundancy elimination can be considered reasonable. Indeed, in our experimental evaluation, redundancy checks typically require one percent of overall algorithm runtime.

6.2 Creating a full learning algorithm

Learning concepts in DLs is a search process. In our proposed learning algorithm, the refinement operator $\rho^{cl}_{\downarrow}$ is used for building the search tree, while a heuristic decides which nodes to expand. As mentioned before, we want to tackle the infinity of the operator by considering only refinements up to some length n at a given time. We call n the *horizontal expansion* of a node in the search tree. It is a node specific upper bound on the length of child concepts, which can be increased dynamically by the algorithm during the learning process.

To deal with this, we formally define a *node* in a search tree to be a triple (C, n, b), where C is a concept, $n \in \mathbb{N}$ is the horizontal expansion, and $b \in \{\text{true}, \text{false}\}$ is a boolean marker for the redundancy of a node.

To define a search heuristic for our learning algorithm, we need some notions to be able to express what we consider a good node for expansion. Similarly to existing ILP systems, we use the learning algorithm parameter *noise*, bounding the minimum acceptable training

[8] More precisely: For each concept there is at most one representative of the equivalence class of weakly equal concepts in the search tree which is evaluated.

set accuracy of the learned definition. $(1 - noise)$ is the lowest accuracy a concept needs to have to be considered a solution of a learning problem.

The search heuristics selects the node with the highest score in the search tree at a given time, where the score of a node is defined as follows:

Definition 14 (Score) Let $N = (C, n, b)$ be a node and `Target` the target concept. We introduce the following notions:

$$accuracy(C) = 1 - \frac{up + cn}{|E|}$$

$$acc\_gain(N) = accuracy(C) - accuracy(C')$$

where C' is the concept in the parent of N

$up = |\{e \mid e \in E^+, \mathcal{K} \cup \{\texttt{Target} \equiv C\} \not\models e\}|$ (uncovered positives)

$cn = |\{e \mid e \in E^-, \mathcal{K} \cup \{\texttt{Target} \equiv C\} \models e\}|$ (covered negatives)

$E = E^+ \cup E^-$

If $up > \lfloor noise \cdot |E| \rfloor$, then the node is *too weak*, i.e. it is not a solution candidate and will never be expanded.

If the node is not too weak, then its score is defined as follows:

$$score(N) = accuracy(C) + \alpha \cdot acc\_gain(N) - \beta \cdot n \quad (\alpha \geq 0,\ \beta > 0)$$

By default, we choose $\alpha = 0.5$ and $\beta = 0.02$. The heuristic uses accuracy as main criterion. Accuracy gain, controlled by α, is incorporated, because those concepts having led to an improvement in accuracy are more likely to be significant refinements towards a solution. As a third criterion, controlled by β, we bias the search towards shorter concepts and less explored areas of the search space. Using horizontal expansion instead of concept length makes the algorithm more flexible in searching less explored areas of the search space and avoids that it gets stuck on concepts with high accuracy and accuracy gain. The score function can be defined independently of the core learning algorithm.

We have now introduced all necessary notions to specify the complete learning algorithm, given in Algorithm 1. *checkRed* is the redundancy check function and *transform* the function to transform a concept to ordered negation normal form as described in Sect. 6.1.

As we can see, the learning algorithm performs a top down refinement operator driven heuristic search. The main difference to other learning algorithms of this kind is the replacement of a full node expansion by a one step horizontal expansion and the use of a redundancy check procedure.

Apart from knowledge representation, another difference to many ILP programs is the search strategy: Often, ILP tools perform a clause by clause set covering approach to construct a solution stepwise. In DL-Learner, each concept represents a full solution, which is related to single predicate theory learning (Bratko 1999) in ILP. A benefit is that there is no risk in performing possibly suboptimal choices and it is often possible to learn shorter solutions. However, it also leads to a higher runtime and memory consumption. To counteract this, a divide and conquer strategy as extension of Algorithm 1 can be activated in DL-Learner. It restricts the set of nodes which are candidates for expansion to a set of fixed size in regular time intervals. By default, the candidate set is restricted to the 20 most promising nodes each 300 seconds. Those concepts are selected according to their accuracy with a

Algorithm 1: learning algorithm

Input: background knowledge, examples E, *noise* in [0,1]
ST (search tree) is set to the tree consisting only of the root node $(\top, 0, \mathit{false})$
while ST *does not contain a node with* $q < \lfloor \mathit{noise} \cdot |E| \rfloor$ **do**
  choose a node $N = (C, n, b)$ with highest score in ST
  expand N up to length $n+1$, i.e.:
  begin
    add all nodes $(D, n, \mathit{checkRed}(ST, D))$ with $D \in \mathit{transform}(\rho^{cl}_{\downarrow}(C))$ and $|D| = n+1$ as children of N
    evaluate created non-redundant nodes
    change N to $(C, n+1, b)$
  end
Return found concepts in ST

bias towards short concepts with high accuracy on positive examples, since those concepts are more likely to improve in a downward refinement algorithm. We omit the details of this process for brevity. It can be used as a tradeoff between performance and the risk to make suboptimal decisions.

Correctness of the algorithm can be shown:

Proposition 15 (Correctness) *If a learning problem has a solution in $\mathcal{ALC}$, then Algorithm 1 terminates and computes a correct solution of the learning problem.*

Proof Assume, there is a solution C (which is an $\mathcal{ALC}$ concept) of a learning problem. By the weak completeness of $\rho^{cl}_{\downarrow}$, we know that there is a concept D with $D \equiv C$ and $D \in \rho^*(\top)$, i.e. $\rho^{cl}_{\downarrow}$ allows a refinement chain $\top \rightsquigarrow D_1 \rightsquigarrow D_2 \rightsquigarrow \cdots \rightsquigarrow D_n = D$. We have already shown in Lemma 5 that $\rho^{cl}_{\downarrow}$ does not reduce length, i.e. all concepts in this chain have at most the length of D. This means that the *score* of each node is higher than $-|D|$ where $|D|$ is the length of D. Because β in the score function is higher than 0, any node with sufficiently high horizontal expansion has a score lower than $-|D|$. As a consequence, all nodes in our chain will eventually be expanded sufficiently often to refine to its successor in the chain above, i.e. eventually D will be reached unless the algorithm terminates with a different solution beforehand. In both cases the proposition is satisfied. □

6.3 Optimisations

In this section improvements of ρ and $\rho^{cl}_{\downarrow}$ will be presented.

Using $\exists r.(C \sqcup D) \equiv \exists r.C \sqcup \exists r.D$ and $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ The equivalences $\exists r.(C \sqcup D) \equiv \exists r.C \sqcup \exists r.D$ and $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ can be used to modify ρ without losing weak completeness.

Disjunctions in ρ are only introduced in refinements of the top concept. The only existential value restrictions in these disjunctions are of the form $\exists r.\top$ for $r \in N_R$. The equivalence $\exists r.(C \sqcup D) \equiv \exists r.C \sqcup \exists r.D$ says that it is not necessary to allow several disjuncts of the form $\exists r.\top$ for a fixed role r, because we can always reach an equivalent concept by only introducing it once. Therefore, we can restrict ρ to produce $\exists r.\top$ only at most once per role as element of the disjunction in the refinement of the top concept, without losing completeness.

ρ allows to refine a concept C by extending it conjunctively. If C is of the form $\forall r.D$ or of the form $C_1 \sqcap \cdots \sqcap \forall r.D \sqcap \cdots \sqcap C_n$, then we can restrict ρ to disallow adding an element of the form $\forall r.E$ (E is an arbitrary $\mathcal{ALC}$ concept). Again, the resulting operator is still complete.

Similar checks can be done in the refinement steps which refine roles. By using the equalities $\exists r.(C \sqcup D) \equiv \exists r.C \sqcup \exists r.D$ and $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as described, we have reduced the number of possible refinements, but preserved completeness.

Configurable target language One of the factors determining whether a learning algorithm will be successful for a given task is whether its target language is suitable. Our algorithm can be configured to some extent in this respect by adapting the used refinement operator. In particular, DL-Learner allows to ignore a specified set of concepts and roles (or conversely use only a specified set of concepts and roles) when trying to solve a problem. It can also be configured to use or ignore the $\exists$ and $\forall$ concept constructors, negation, cardinality restrictions, double concrete roles, boolean concrete roles. These options can be used to incorporate additional knowledge about a problem in the learning algorithm to narrow the search space. Note, that all experiments reported in Sect. 7 were run without such restrictions.

7 Evaluation

7.1 Comparison with other algorithms based on description logics

We illustrate our algorithm using Michalski's trains (Michalski 1980) as a simple example. The data describes different features of trains, e.g. which cars are appended to a train, whether they are short or long, closed or open, jagged or not, which shapes they contain and how many of them. The positive examples are the trains on the left in Fig. 2 and the negative examples are the trains on the right. Thus, the task of the learner is to find characteristics of all the left trains, which none of the right trains has. The learning algorithm first explores the concepts $\top$ and then `Train`, which cover all examples. Other atomic concepts are too weak to be considered for further exploration. The exploration of the top concept up to a horizontal expansion of 3 leads to $\exists$`hasCar`.$\top$, which is then expanded to $\exists$`hasCar.Closed`. This covers all positives and two negatives. The heuristic later picks this node and extends it up to a horizontal expansion of 5 to $\exists$`hasCar.(Closed` $\sqcap$ `Short)`, which is a possible (and shortest) solution for the problem.

Doubtless, there is a lack of evaluation standards in ontology learning from examples. In order to overcome this problem, we converted the background knowledge of several existing learning problems to OWL ontologies. Besides the described train problem, we also investigated the problems of learning family relationships from FORTE (Richards and Mooney 1995), learning poker hands, and understanding the moral reasoning of humans. The two

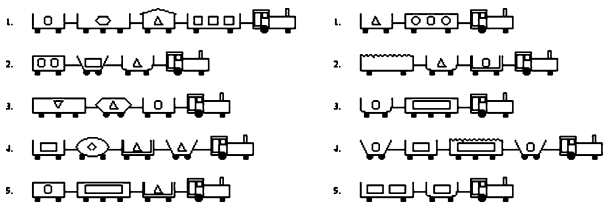

Fig. 2 The Michalski trains problem: positive examples are on the *left*, negative examples are on the *right*

latter examples were taken from the UCI Machine Learning repository.[9] For the poker example, we defined two goals: learning the definition of a pair and of a straight. Similarly, the moral reasoner examples were divided into two learning tasks: the original one, where the intended solution is quite short, and a problem where we removed an important intermediate concept, such that the smallest possible solution became more complex. For the FORTE family data set, we defined the problem of learning the definition of an uncle (originally defined in Lehmann 2007), where one possible solution is:

$$\texttt{Male} \sqcap (\exists \texttt{sibling}.\exists \texttt{parent}.\top \sqcup \exists \texttt{married}.\exists \texttt{sibling}.\exists \texttt{parent}.\top)$$

The poker example has medium size, but is not very complex with respect to its terminology. The moral reasoner, however, is an expressive ontology, which we derived from a theory given as a logic program. We designed the FORTE problem to be slightly more difficult for our algorithm, because no simple and short solutions exist. Overall, the solutions of the examples cover a range of different concept constructors and are of varying length and complexity. Section 7.2 will describe a current ILP benchmark, the carcinogenesis problem, in order to compare the described algorithm with inductive learning algorithms which are not based on description logics.

As a reasoner we used Pellet,[10] which was connected to the learning problem via the DIG reasoner interface[11] for YinYang (Iannone et al. 2007) and the OWL API[12] interface for DL-Learner (because DIG does not support asking for domains and ranges), on a 2.2 GHz Dual CPU machine with 2 GB RAM. We compared our results with those from YinYang and other algorithms we have implemented within the DL-Learner framework. In particular, we compared with a hybrid algorithm using so called genetic refinement operators (Lehmann 2007), i.e. adapted refinement operators within a Genetic Programming (GP) framework. For reference we also compared with a standard GP learning algorithm, which has been applied to the learning problem in description logics and is also included in DL-Learner. For all algorithms, we used five fold cross validation. We are not aware of other systems available for comparison. The system in Cohen and Hirsh (1994) is no longer available and the approach in Badea and Nienhuys-Cheng (2000) was not fully implemented.

For YinYang we used the same settings as in all examples included in its release and for the refinement approach we used the standard settings. For the GP algorithms, we chose a fixed number of 50 generations with a population of 500 individuals. A generational algorithm with rank selection and activated elitism was used. The algorithms were initialised using the ramped-half-and-half method with maximum depth 6. For the standard GP algorithm, a crossover probability of 80 percent and 2 percent mutation probability was set. The hybrid approach was configured to 65 percent genetic refinement, 20 percent crossover, and 2 percent mutation. In both cases, the fitness measure parameter α was adjusted to a low value (0.002), such that the correct solutions have a sufficiently high value in the GP fitness function. The settings are similar to those found to be suitable in Lehmann (2007), where population size is varied between 100 and 700. Naturally, a GP algorithm always allows to increase the number of invested resources and varying the population size is one of the ways to do this. We finally picked a population size of 500 for our evaluation, since in example

[9] http://www.ics.uci.edu/~mlearn/MLRepository.html.

[10] http://pellet.owldl.com.

[11] http://dl.kr.org/dig/.

[12] http://owlapi.sf.net/.

Table 3 Evaluation results for various DL-Learner algorithms and YinYang

Problem	YinYang			
	Time (s)		Length	Correct (%)
trains	0.2 ± 0.1		8.1 ± 1.5	100.0 ± 0.0
moral I	28.8 ± 12.1		69.0 ± 16.1	50.0 ± 21.7
moral II	32.6 ± 9.5		70.7 ± 21.8	62.5 ± 28.0
poker I	7.2 ± 0.8		43.2 ± 12.1	100.0 ± 0.0
poker II	–		–	–
forte	26.4 ± 9.4		22.1 ± 12.0	90.0 ± 5.6
Problem	DL-Learner Refinement			
	Time (s)	Time 2 (s)	Length	Correct (%)
trains	0.8 ± 0.3	0.1 ± 0.0	5.0 ± 0.0	100.0 ± 0.0
moral I	2.8 ± 0.3	0.1 ± 0.0	8.0 ± 0.0	97.8 ± 5.0
moral II	2.7 ± 0.4	0.1 ± 0.0	8.0 ± 0.0	97.8 ± 5.0
poker I	3.4 ± 0.1	0.0 ± 0.0	5.0 ± 0.0	100.0 ± 0.0
poker II	19.7 ± 10.6	1.5 ± 0.1	11.0 ± 0.0	100.0 ± 0.0
forte	13.4 ± 1.8	0.3 ± 0.1	13.4 ± 0.9	98.9 ± 2.5
Problem	DL-Learner GP			
	Time (s)		Length	Correct (%)
trains	33.4 ± 3.1		3.4 ± 0.9	60.0 ± 41.8
moral I	28.9 ± 1.4		1.0 ± 0.0	86.1 ± 10.0
moral II	31.0 ± 5.9		1.6 ± 0.9	62.8 ± 9.1
poker I	465.9 ± 261.3		3.4 ± 2.2	84.0 ± 21.9
poker II	283.3 ± 12.7		1.0 ± 0.0	92.7 ± 0.3
forte	235.9 ± 74.6		3.0 ± 0.0	88.1 ± 8.0
Problem	DL-Learner hybrid GP			
	Time (s)		Length	Correct (%)
trains	10.5 ± 1.1		4.6 ± 0.9	80.0 ± 44.7
moral I	139.9 ± 10.0		3.0 ± 0.0	100 ± 0.0
moral II	118.7 ± 26.8		2.8 ± 1.1	71.9 ± 13.1
poker I	709.1 ± 60.7		5.0 ± 0.0	100.0 ± 0.0
poker II	1054.1 ± 36.5		1.0 ± 0.0	92.7 ± 0.3
forte	285.0 ± 25.6		3.0 ± 0.0	88.1 ± 8.0

runs it seemed to deliver a good tradeoff between runtime and cross validation accuracy. Overall, the runtime of the GP algorithms is often much higher than those of DL-Learner and YinYang in this setting.

Table 3 summarises the results we obtained. As a statistical significance test, we used a t-test with 95% confidence interval. In all cases, our implementation was able to learn a correct definition on the training set, which in most cases also was correct on the testing set. YinYang could not handle the second poker problem (it produces an error after trying to

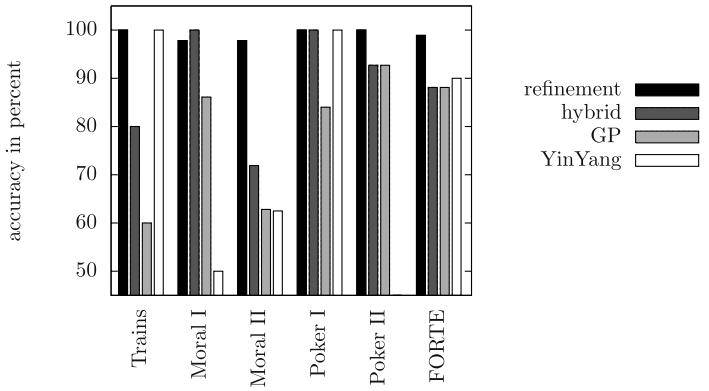

Fig. 3 Accuracy comparison: The refinement approach has a statistically significantly higher accuracy than all others on the complex moral reasoner, poker II, and forte problems, while none of the other algorithms performs statistically significantly better on any of the other learning problems

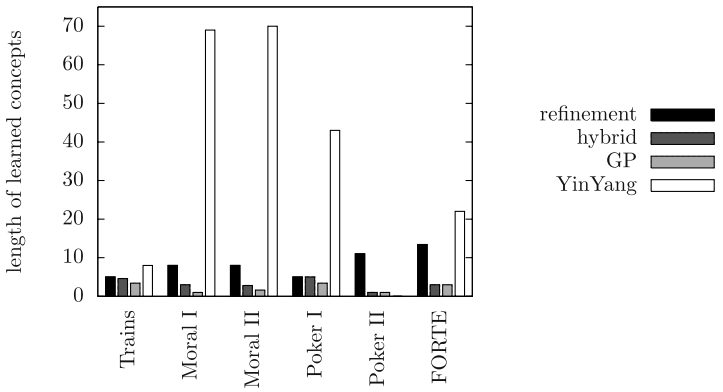

Fig. 4 Comparison of the length of the learned concepts: YinYang produces longer concepts, which were in general more difficult to read for humans, than the refinement approach with high statistic significance. The genetic programming approaches, in particular the standard variant, are often unable to solve the more complex problems in reasonable time

compute most specific concepts). Similarly, the FORTE problem could only be handled after removing certain examples (4 out of 86). Figure 3 visualises the obtained cross validation accuracies. The refinement approach has a statistically significantly higher accuracy than all others on the complex moral reasoner, poker II, and forte problems, while none of the other algorithms performs statistically significantly better on any of the other learning problems. The hybrid GP approach generally performed at least as good as the standard GP approach and often it was (statistically significantly) better.

As another interesting criterion, we recorded the length of learned concepts (see Fig. 4). A notable observation is that YinYang produces longer concepts than the refinement approach with high statistic significance. In most cases, this rendered those concepts hard to read for humans, which is a disadvantage for symbolic classifiers. In contrast, the genetic programming approaches often produce short concepts. The standard GP algorithm usu-

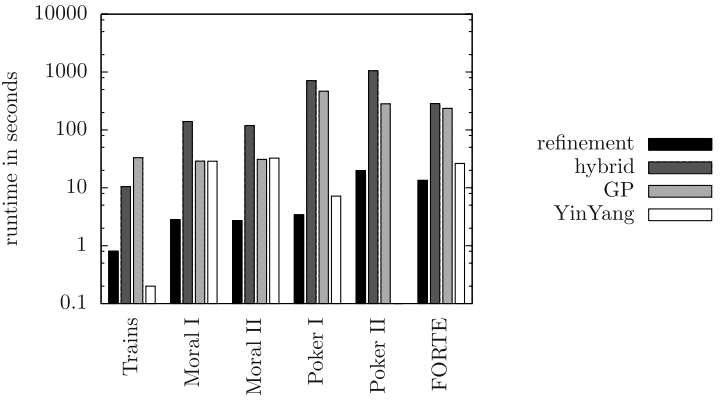

Fig. 5 Runtime comparison: The GP algorithms generally have the highest runtimes unless we trade accuracy for runtime. The refinement operator is statistically significantly faster than all other approaches for all problems except trains

ally learned such short solutions, because it is often unable to find more accurate longer definitions. The hybrid approach is more likely to find complex definitions, e.g. it sometimes found one of the possible solutions (`Severity_harm` $\sqcap \neg$ `Benefit_victim` $\sqcap$ (`Responsible` $\sqcup$ `Vicarious`)) of the second moral reasoner problem.

Figure 5 compares the runtime of the different algorithms (note the logarithmic scale). Overall, the GP algorithms have a higher runtime. As mentioned before, they can be parametrised to have a shorter runtime if we are willing to accept a decline in accuracy. In general, the refinement operator is statistically significantly faster than all other approaches for all problems except trains.

For the refinement approach, we included another column "time 2" in Table 3. These are the learning times when employing a built-in approximate OWL reasoner instead of Pellet. It works by performing a number of initial queries to a standard OWL reasoner and then keeps the results in memory to efficiently answer all other requests (without correctness and completeness guarantee w.r.t standard semantics). It uses a form of closed world reasoning, which solves open world issues first raised in Badea and Nienhuys-Cheng (2000). Using the open world assumption, an object a is not instance of $\forall r.C$ even if all *known* r-fillers of a are instances of C, which makes it hard to learn concepts like $\forall$`hasChild.Male` (only male children). So the advantage of using this reasoner is two-fold: it has much better performance and it uses closed world reasoning, which is often desired during the learning process.

Although using closed world reasoning can change the node score and therefore influences the learning process, it did not impact on the cross validation accuracy in the presented examples except for a single fold in the poker II learning problem, where it stopped after finding a shorter concept covering all training set examples, and therefore had an overall cross validation accuracy of 96.2% instead of the 100% shown in the table.

All learning problems are available at the DL-Learner subversion repository[13] and a script is provided to reproduce the results presented here.

[13] Browsable e.g. via http://dl-learner.svn.sourceforge.net/viewvc/dl-learner/.

7.2 Comparison with other ILP approaches

To compare DL-Learner with other ILP approaches and to apply it in a realistic scenario, we choose the problem of predicting carcinogenesis. The aim of this task is to predict whether a chemical compound causes cancer given its structure and the results of bio-assays. This has been recognised as an important research area: "Obtaining accurate structural alerts for the causes of chemical cancers is a problem of great scientific and humanitarian value" (Srinivasan et al. 1997). Although this is one of the most well-researched problems in Machine Learning, it is still important and challenging.

One of the problems we have to face when benchmarking DL-Learner is of course that many of the problems are available in a Prolog-like syntax. The first step to apply DL-Learner to the carcinogenesis problem was to convert the original data[14] into an OWL ontology. To do this, we extended DL-Learner with a Prolog parser and wrote a mapping script to convert the carcinogenesis files into OWL. As mentioned previously, OWL (description logics) and logic programs have incomparable expressivity. It is sometimes impossible and often not trivial to convert between both representations. For carcinogenesis such a mapping is possible, but required at least a superficial understanding of the domain. The mapping script we used and the resulting ontology are both freely available at the DL-Learner download page.[15] During the transformation process almost no knowledge was lost or added. The resulting ontology contains 142 atomic concepts, 19 roles and datatypes, 22373 objects, and more than 74000 facts.

We used the approximate reasoner introduced in the previous subsection. Standard OWL reasoners turned out to be much too slow in answering reasoning requests for this ontology.[16] Furthermore, we also enabled all extensions presented in Sect. 5.4. Again, all tests were run on a 2.2 GHz Dual core machine with 4 GB RAM.

The most sensible parameter of DL-Learner in the case of carcinogenesis prediction is noise (bounding the minimum acceptable training set accuracy of the learned definition). To find an appropriate setting for the noise parameter, we divided the available examples into two sets. In a first phase, 30% of the 337 examples were used to find the noise parameter value. This was done by starting from a noise value of 50% and descending in one percent steps. We measured the ten fold cross validation accuracy for each of those values. It turned out that a noise value of 30% has the highest accuracy (71.8%). In a second phase, we used this parameter to measure the ten fold cross validation accuracy on the second set containing 70% of all examples. This lead to an accuracy of 67.7% as shown in Table 4 together with results of other approaches using the same background knowledge (many of those use Aleph, a state-of-the-art Inductive Logic Programming system, as their basis).

For all approaches, where the standard deviation was given in the articles, we calculated whether the difference in accuracy is statistically significant using a t-test. We obtained P values of 0.0508 vs. Aleph DTD 0.7, 0.0231 vs. Aleph RRR 0.9, 0.0206 vs. Aleph DTD 0.9, and 0.0107 vs. Aleph RRR 0.7. Values ≤ 0.05 (3 out of 4 in this case) are statistically significant with a standard confidence interval of 95%.

The average runtime for a noise value of 30% was 950 seconds with a concept length of 20 when measured on all examples. We will briefly compare this with the other approaches in Table 4.

[14] http://web2.comlab.ox.ac.uk/oucl/research/areas/machlearn/cancer.html.

[15] http://sourceforge.net/projects/dl-learner.

[16] While scalability of OWL reasoning systems has improved substantially in the recent past, OWL and related DLs are inherently difficult to reason with, as their worst-case complexity classes are ExpTime or worse.

Table 4 Overview of the accuracy of different ILP approaches applied to the carcinogenesis prediction problem. Many of those are improvements or different settings of Aleph. Accuracy and standard deviation refer to values obtained through 10 fold cross validation. Legend: DTD = Deterministic Top Down, RRR = Randomized Rapid Restarts, +, ∘, − stand for good, moderate, and bad readability of learning results (see text)

Approach/tool	Accuracy	Readability	Reference
DL-Learner	67.7% ± 11.3%	+	
Aleph Ensembles	59.0% to 64.5%	−	(Dutra et al. 2003)
Boosted Weak ILP	61.1%	−	(Jiang and Colton 2006)
Weak ILP	58.7%	−	(Jiang and Colton 2006)
Aleph DTD 0.7	57.9% ± 9.8%	∘	(Železný et al. 2003)
Aleph RRR 0.9	57.6% ± 6.4%	∘	(Železný et al. 2003)
Aleph DTD 0.9	56.2% ± 9.0%	∘	(Železný et al. 2003)
Aleph RRR 0.7	54.8% ± 9.0%	∘	(Železný et al. 2003)

Dutra et al. (2003) is a bagging approach combining 1 to 100 hypotheses generated by Aleph. The results vary from 59.0% to 64.5% accuracy depending on the choosen ensemble size. We believe that a combination of a high number of hypotheses scores lower on human interpretability, in particular since the concepts provided by DL-Learner are quite compact. The results themselves were computed on Condor,[17] a high throughput computing system and consumed 53580 CPU hours including 3 other experiments apart from carcinogenesis. Even taking the different ensemble sizes and parameter optimisation phases in this experiment into account, our approach seems to be competitive.

Jiang and Colton (2006) did not record the runtime of experiments. The system uses a boosting approach with 50 to 100 base classifiers based on WeakILP. Using a similar length measure than the one we defined for DL concepts, i.e. counting all logical symbols, the summed length is approximately 1000 compared to 13.4 in our case. Hence, we also consider this approach to produce less readable results.

Železný et al. (2003) reports runtimes of 6 to 7 hours for the two DTD approaches on a 1.5 GHz machine with 512 MB RAM. The RRR approaches are much faster and need only 26 minutes. Regarding readability, the length of learned programs is about 100. Hence, they are much shorter than those of the two other approaches, but can still be considered harder to interpret than the results of DL-Learner.

To illustrate the influence of the noise parameter, we additionally measured ten fold cross validation accuracy, runtime, and concept length on all examples with different noise values. The results are shown in Table 5. Since the noise parameter acts as a termination criterion, we observe, as expected, that lower noise values lead to significant increases in runtime. It becomes increasingly computationally expensive for the learning algorithm to find concepts satisfying the termination criterion and those concepts are usually also more complex as evident from the last column of the table. Therefore, setting the noise values too low can also lead to learning unnecessary complex concepts as shown in Fig. 6.

[17] http://www.cs.wisc.edu/condor/.

Table 5 The influence of the noise parameter on ten fold cross validation accuracy, runtime, and length of learned concepts. We see that for lower noise values it becomes increasingly hard to satisfy the termination criterion (hence the increase in runtime) and the learned concepts are longer and more complex. The maximum accuracy is reached for 32% noise and stays on a similar level afterwards, which indicates that the additional structures in those longer concepts do not greatly affect classification results. Note that the runtime does not include the time to load the knowledge base into the reasoner and prepare it, which takes additional 36 seconds on our test machine

Noise(%)	Accuracy(%)	Runtime(s)	Length
40	62.9 ± 8.6	0.6 ± 0.6	4.9 ± 1.4
39	62.9 ± 8.6	0.6 ± 0.7	4.9 ± 1.4
38	65.9 ± 8.3	1.5 ± 0.2	7.0 ± 0.0
37	65.9 ± 8.3	5.2 ± 7.8	7.6 ± 1.3
36	65.9 ± 8.3	6.9 ± 8.5	7.9 ± 1.4
35	65.9 ± 8.3	12.7 ± 9.6	8.8 ± 1.6
34	64.4 ± 6.6	31.6 ± 25.7	9.7 ± 0.7
33	64.7 ± 6.5	73.6 ± 88.4	9.8 ± 0.9
32	67.4 ± 7.9	160.0 ± 197.2	10.7 ± 3.1
31	66.4 ± 7.5	426.9 ± 324.2	14.1 ± 3.7
30	65.9 ± 8.7	843.5 ± 538.1	17.9 ± 4.5
29	66.8 ± 8.1	1613.9 ± 922.3	23.2 ± 5.0
28	66.5 ± 9.0	3158.3 ± 1680.8	29.6 ± 5.8

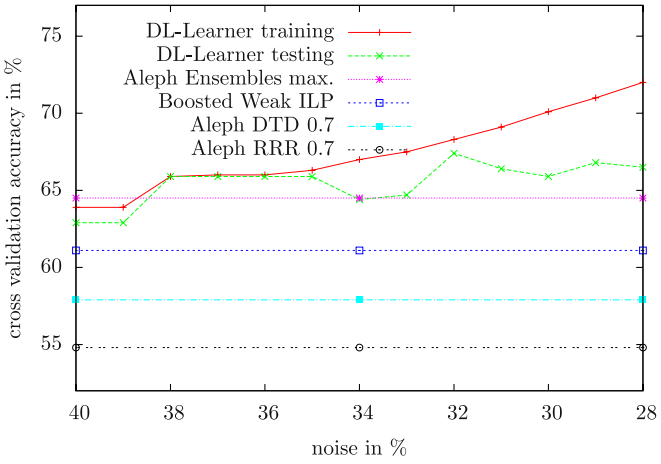

Fig. 6 Illustration of ten fold cross validation accuracies of DL-Learner on the carcinogenesis benchmark for different noise values. For lower noise values the difference between training and testing accuracy (averaged over all ten folds) becomes larger, which can be interpreted as a sign of overfitting, i.e. the learned concepts are unnecessarily complex. As a reference, we included the accuracies of other tools from Table 4 as *horizontal lines*

As an example of a learned concept, the following definition was one of the more complex concepts learned with noise = 28%:

$$\begin{aligned}(\text{Compound} \sqcap \neg \exists \text{hasAtom}.(&\text{Nitrogen-35} \sqcup \text{Phosphorus-60} \\ &\sqcup \text{Phosphorus-61} \sqcup \text{Titanium-134}) \\ \sqcap (\geq 3 \text{hasStructure}.(&\text{Halide} \sqcap \neg \text{Halide10}) \\ &\sqcup (\text{amesTestPositive} = \text{true} \sqcap \geq 5 \text{hasBond}.(\neg \text{Bond-7}))))\end{aligned}$$

This can be phrased in natural language as:

A chemical compound is carcinogenic iff . . .
. . . it does not contain a Nitrogen-35, Phosphorus-60, Phosphorus-61, or Titanium-134 atom
. . . and it has at least three Halide—excluding Halide10—structures or the ames test of the compound is positive and there are at least five atom bonds which are not of bond type 7.

Overall, the presented approach is able to learn accurate and short concepts with a reasonably low number of expensive reasoner requests. Note that all the approaches are able to learn in a very expressive language with arbitrarily nested structures, as can be seen in the concept above. Learning many levels of structure has recently been identified as a key issue for structured Machine Learning (Dietterich et al. 2008), and our work provides a clear advance on this front.

The evaluations show that our approach is competitive with state-of-the-art ILP systems when the approximate reasoning technique is used.

8 Strengths and limitations of the described approach

This section summarizes the advantages and disadvantages of the presented algorithm. It is organised along the criteria of applicability, accuracy, readability, scalability, and usability. Apart from analysing strengths and limitations of the concrete approach, we also want to draw a more general picture on why learning in DLs is a promising line of research.

Applicability The Semantic Web is rapidly growing[18] and contains knowledge from diverse areas such as science, music, people, books, reviews, places, politics, products, software, social networks, as well as upper and general ontologies. The underlying technologies, sometimes called *Semantic Technologies*, are currently starting to create substantial industrial impact in application scenarios on and off the web, including knowledge management, expert systems, web services, e-commerce, e-collaboration, etc. Data exchange and integration is central to these technologies, which thus hinge centrally on the use of suitable knowledge representation formalisms. Consequently, it is important to adhere to established standards, foremost those established by the World Wide Web Consortium (W3C). Since 2004, the Web Ontology Language OWL, which is based on description logics (DLs), has been the W3C-recommended standard for Semantic Web knowledge representation and has

[18]For instance, the semantic index Sindice (http://sindice.com/) grows steadily and now lists more than 10 billion entities from more than 100 million web pages.

been a key to the growth of the Semantic Web. Being able to apply inductive learning on this data and to use OWL/DLs themselves as results of a learning algorithm widens the possibilities for ILP research and practice since it opens up Semantic Web as field of application.

It could be argued that ILP systems based on logic programs can be used to achieve this task. However, as discussed previously, OWL ontologies and logic programs are incomparable with respect to their expressiveness, i.e. there are OWL ontologies not expressible in Horn logic and vice versa. This means that the algorithm cannot be applied to all scenarios where Horn logic ILP programs can be used, which is both a strength and a limitation of the described approach. One restriction are predicates with arity greater two. Since concepts in description logics correspond to predicates of arity one and roles correspond to predicates of arity two, it is not straightforward to express predicates with higher arity. However, in many cases, e.g. the benchmarks in Sect. 7, this can still be done but usually requires a human expert. Even in such cases, it often turned out that knowledge can be represented in a human friendlier and more readable way in OWL/description logics.

It should be noted that in principle each learning problem in DLs can be solved by ILP systems based on first order logic, since DLs are a fragment of first order logic. However, it can of course be inefficient to do this due to the larger search space and higher complexity of reasoning. Badea and Nienhuys-Cheng (2000) has discussed why ILP systems may not be appropriate to learn DL concepts even in those cases where they can be used in principle. The paper analyses this for the case of using prenex conjunctive normal form (PCNF) and states that the main problems are that 1.) a conversion to PCNF can lead to an exponential blowup in knowledge base size and 2.) many formulae in PCNF do not have a counterpart in Description Logics, i.e. they are too fine grained. Similar arguments apply when using Horn logic or first order logic. On the other hand, using systems for less expressive formalisms cannot make use of all the carefully selected features of OWL.

OWL and Description Logics also enable *new application tasks*. One such task is ontology engineering, in particular suggesting definitions and equivalence axioms in knowledge bases based on instance data. We pursue this line of research by developing plugins for the popular Protege[19] and OntoWiki[20] ontology editors. There have been heated discussions in the recent past on the use of logic programming versus description logics in ontology engineering, see e.g. de Bruijn et al. (2005), Patel-Schneider and Horrocks (2007) and Krötzsch et al. (2008a, 2008b) for recent developments. However the recently rapidly growing popularity of the Web Ontology Language OWL seems to indicate that some of their distinctive features make it a viable and perhaps even superior alternative to logic programs in many application domains. This can in part be attributed to the fact that DLs restrict the modeller more severely in his use of the modelling language: Horn logic is Turing complete (Šebelík and Štěpánek 1982) (and can, e.g. in the form of Prolog, even be used as a programming language), while DLs are usually decidable.

Accuracy We have shown to be more accurate than other DL learning systems and we claim to be more accurate than other general ILP tools for some learning problems. We have shown this for carcinogenesis, where DL-Learner was able to achieve statistically significantly higher accuracy than some state-of-the-art ILP systems. It is well-known that the choice of the target language is a critical one with respect to the accuracy and efficiency of learning algorithms. Hence, it is natural that DL concepts are more appropriate than

[19] http://protege.stanford.edu/.

[20] http://ontowiki.net.

other formalisms for a subset of learning problems. This particularly applies to domains, where ontologies are already widely used, e.g. the life sciences (Rector and Brandt 2008; Belleau et al. 2008).

Readability As shown in Sect. 7, DL-Learner has a bias towards learning compact, readable concepts. We have shown that there are often orders of magnitude difference with respect to the size of the offered solutions in the carcinogenesis problem. Hence, we do consider readability to be a strength of our approach. Furthermore, DLs lend themselves easily to conversion between (controlled) natural language and formal language, see Schwitter et al. (2008), Völker et al. (2007) for references. This helps to close the gap between ontology engineers and domain experts.

Scalability Our experiments indicate that the presented approach can work efficiently with knowledge bases containing around 100.000 axioms—of course depending on the complexity of those axioms. This is sufficient for many realistic application scenarios. Furthermore, we also extended DL-Learner by a component, which allows to apply it to very large knowledge bases (Hellmann et al. 2009) by selecting relevant knowledge fragments in a preprocessing step. We applied the procedure to DBpedia (Auer et al. 2008), containing more than 200 million axioms, and other large knowledge bases.

Usability DL-Learner requires only a minimal set of parameters to work well. Those parameters are the used background knowledge bases (which can be more than one), the positive and negative examples, and a termination criterion (e.g. minimum accuracy through the noise parameter). As the approach is an anytime-algorithm, bounding its maximum runtime can be convenient in systems which need to process several learning problems reliably without the risk of using too many resources. Also note that other tools like Aleph often require so called mode declarations in order to work efficiently by restricting the search space. These restrictions are already present in DL/OWL knowledge bases through domain and ranges of roles. They are used by the refinement operator automatically, which makes it easier to apply DL-Learner without the need to specify further restrictions (which requires knowledge about the domain at hand).

Summary In summary, we argue that learning in DLs is limited in that not all typical ILP problems can be solved. However, it is also apparent that it can widen the scope of ILP to new application areas and tasks, in particular in the context of Semantic Web applications and application development which hinges critically on the employed knowledge representation formalisms. We have shown that DL-Learner is competitive with respect to accuracy and scalability with state-of-the-art ILP systems. We also claim that the provided solutions are more readable and that DL-Learner is easy to use.

9 Related work

Related work can essentially be divided in two categories. The first is research which is directly connected to learning in description logics. The second is research about refinement operators in general, often connected with the learning of logic programs. We will describe both in turn.

In Badea and Nienhuys-Cheng (2000) a refinement operator for $\mathcal{ALER}$ has been designed to obtain a top-down learning algorithm for this language. Properties of refinement

operators in this language were discussed and some claims were made, but a full formal analysis was not performed. They also investigate some theoretical properties of refinement operators. As we have done with the design of ρ, they favour the use of a downward refinement operator to enable a top-down search. The authors use $\mathcal{ALER}$ *normal form*, which is easier to handle than negation normal form, because $\mathcal{ALER}$ is not closed under boolean operations. As a consequence, they obtain a simpler refinement operator for which it is not clear how it could be extended to more expressive DLs. Our operator, in contrast, lends itself much more easily to such extensions. We also deal quite differently with infinity, we show how the subsumption hierarchy of atomic concepts and roles can be used, use domain and range of roles to structure the search, and we describe how redundancy can be avoided efficiently. Moreover, our theoretical results are more general, i.e. covering more description languages and property combinations. In contrast to Badea and Nienhuys-Cheng (2000), we provided proofs, which were not available on request by the authors, and refuted one of their results. As mentioned before, the algorithm in Badea and Nienhuys-Cheng (2000) was not implemented, which is why we could not assess its performance on our learning examples.

In the papers (Esposito et al. 2004; Iannone and Palmisano 2005) and (Iannone et al. 2007), learning algorithms for description logics, in particular for the language $\mathcal{ALC}$ were created, which also make use of refinement operators—however, not as centrally as in our approach. The core idea of those algorithms is blame assignment, i.e. to find and remove those parts of a concept responsible for classification errors. In particular, Iannone et al. (2007) described how to apply the learning problem for classifying scientific papers. Instead of using the classical approach of combining refinement operators with a search heuristic, a different approach is taken therein for solving the learning problem by using approximated MSCs (most specific concepts). A problem of these algorithms is that they tend to produce unnecessarily long concepts. One reason is that MSCs for $\mathcal{ALC}$ and more expressive languages do not exist and hence can only be approximated. Previous work (Cohen et al. 1993; Cohen and Hirsh 1994) in learning in DLs has mostly focused on approaches using least common subsumers, which face this problem to an even larger extent according to their evaluation. In our approach, we also cannot guarantee that we obtain the shortest possible solution of a learning problem. However, the learning algorithm was carefully designed to produce short and readable solutions. The produced solutions will be close to the shortest solution in negation normal form.

Esposito et al. (2004) and Fanizzi et al. (2004) stated that an investigation of the properties of refinement operators in description logics, as we have done in this article, is required for building a theoretical foundation of the research area. In Fanizzi et al. (2004) downward refinement for $\mathcal{ALN}$ was analysed using a clausal representation of description logic concepts. Refinement operators have also been dealt with within hybrid systems. In Lisi and Malerba (2003) ideal refinement for learning $\mathcal{AL}$-log, a language that merges DATALOG and $\mathcal{ALC}$, was investigated. Based on the notion of $\mathcal{B}$-subsumption, an ideal refinement operator was created. In Kietz and Morik (1994), Cohen and Hirsh (1994) learning algorithms for description logics without refinement operators were analysed.

In the area of Inductive Logic Programming considerable efforts have been made to analyse the properties of refinement operators (for a comprehensive treatment, see e.g. Nienhuys-Cheng and de Wolf 1997). In general, applying refinement operators for clauses to solve the learning problem in DLs is usually not a good choice (Badea and Nienhuys-Cheng 2000). However, the theoretical foundations of refinement operators in Horn logics also apply to description logics, which is why we want to mention work in this area here.

A milestone in Machine Learning (Mitchell 1997) in general was the Model Inference System in Shapiro (1991). Shapiro describes how refinement operators can be used to adapt

a hypothesis to a sequence of examples. Afterwards, refinement operators became widely used as a learning method. van der Laag and Nienhuys-Cheng (1994) have found some general properties of refinement operators in quasi-ordered spaces. Nonexistence conditions for ideal refinement operators relating to infinite ascending and descending refinement chains and covers have been developed. This has been used to show that ideal refinement operators for clauses ordered by θ-subsumption do not exist. Unfortunately, we could not make use of these results directly, because proving properties of covers in description logics without using a specific language is likely to be harder than directly proving the results.

Nienhuys-Cheng et al. (1993) discussed refinement for different versions of subsumption, in particular weakenings of logical implication. It was shown in Nienhuys-Cheng et al. (1999) how to extend refinement operators to learn general prenex conjunctive normal form. Perfect operators, i.e. operators which are weakly complete, locally finite, non-redundant, and minimal, were discussed in Badea and Stanciu (1999). Since such operators do not exist for clauses ordered by θ-subsumption (van der Laag and Nienhuys-Cheng 1994), weaker versions of subsumption were considered. This was later extended to theories, i.e. sets of clauses (Fanizzi et al. 2003). A less widely used property of refinement operators, called flexibility, was discussed in Badea (2000). Flexibility essentially means that previous refinements of an operator can influence the choice of the next refinement. The article discusses how flexibility interacts with other properties and how it influences the search process in a learning algorithm.

10 Conclusions and further work

The contribution of this paper is two-fold. On the one hand, we have provided a complete theoretical analysis of possible desirable property combinations for refinement operators for description logics. Indeed we believe to have presented the first thorough treatment of these matters. The results are summarised in Theorems 1 and 2, and in particular, we have shown that ideal refinement operators for expressive description logics cannot exist. On the other hand, we have presented the first centrally refinement operator based learning algorithm for expressive DLs which are closed under boolean operations. The underlying refinement operator is based on our theoretical investigations summarised in Theorem 2, and we have shown formally that our operator satisfies the desirable properties which are achievable. We also showed how the problems of redundancy and infinity can be solved in a satisfactory manner, allowing us to specify a learning algorithm which we proved to be correct. We implemented the algorithm and an evaluation showed that the resulting system is competetive with state-of-the-art systems.

Further work will focus on several aspects, which we believe will broaden ILP application areas:

One part will involve the integration of the learning algorithm in ontology editors, e.g. OntoWiki (Auer et al. 2006), Protégé,[21] or the NeOn toolkit,[22] and the application in ontology acquisition scenarios. Ways need to be investigated how to integrate DL-Learner into the ontology engineering lifecycle, and how to leverage its power for learning from unstructured raw data like text corpora, e.g. by combining it with text mining techniques (Buitelaar et al. 2007; Völker et al. 2007).

[21] http://protege.stanford.edu.

[22] http://www.neon-toolkit.org.

Another direction involves broadening the applicability of the algorithm to very large Semantic Web knowledge bases. Within the Linking Open Data initiative[23] a fast growing network of interconnected openly available OWL knowledge bases, containing billions of facts covering many domains, has been established. Given positive and negative examples, fragments of relevant knowledge can be extracted from those knowledge bases and, ultimately, an algorithm can autonomously retrieve background knowledge for a given learning task. Some preliminary tests along this direction have been performed using DBpedia (Auer et al. 2008).

Acknowledgements This research was partially supported by the Federal Ministry of Education and Research under the SoftWiki project and by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project.

References

Auer, S., Dietzold, S., & Riechert, T. (2006). OntoWiki—a tool for social, semantic collaboration. In *Lecture notes in computer science: Vol. 4273. The semantic web—ISWC 2006, 5th international semantic web conference, ISWC 2006, Athens, GA, USA, November 5–9, 2006, Proceedings* (pp. 736–749). Berlin: Springer.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2008). DBpedia: A nucleus for a web of open data. In *Lecture notes in computer science: Vol. 4825. Proceedings of the 6th international semantic web conference (ISWC)* (pp. 722–735). Berlin: Springer.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (Eds.) (2007). *The description logic handbook: theory, implementation, and applications*. Cambridge: Cambridge University Press.

Badea, L. (2000). Perfect refinement operators can be flexible. In W. Horn (Ed.), *Proceedings of the 14th European conference on artificial intelligence* (pp. 266–270). Amsterdam: IOS Press.

Badea, L., & Nienhuys-Cheng, S.-H. (2000). A refinement operator for description logics. In J. Cussens & A. Frisch (Eds.), *Lecture notes in artificial intelligence: Vol. 1866. Proceedings of the 10th international conference on inductive logic programming* (pp. 40–59). Berlin: Springer.

Badea, L., & Stanciu, M. (1999). Refinement operators can be (weakly) perfect. In S. Džeroski & P. Flach (Eds.), *Lecture notes in artificial intelligence: Vol. 1634. Proceedings of the 9th international workshop on inductive logic programming* (pp. 21–32). Berlin: Springer.

Belleau, F., Tourigny, N., Good, B., & Morissette, J. (2008). Bio2RDF: A semantic web atlas of post genomic knowledge about human and mouse. In A. Bairoch, S. C. Boulakia, & C. Froidevaux (Eds.), *Lecture notes in computer science: Vol. 5109. DILS* (pp. 153–160). Berlin: Springer.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1990). Occam's razor. In J. W. Shavlik & T. G. Dietterich (Eds.), *Readings in machine learning* (pp. 201–204). San Mateo: Morgan Kaufmann.

Brachman, R. J. (1978). *A structural paradigm for representing knowledge*. Technical Report BBN Report 3605, Bolt, Beraneck and Newman, Inc., Cambridge, MA.

Bratko, I. (1999). Refining complete hypotheses in ILP. In S. Džeroski & P. Flach (Eds.), *Lecture notes in artificial intelligence: Vol. 1634. Proceedings of the 9th international workshop on inductive logic programming* (pp. 44–55). Berlin: Springer.

Buitelaar, P., Cimiano, P., & Magnini, B. (Eds.) (2007). *Ontology learning from text: Methods, evaluation and applications. Frontiers in artificial intelligence* (Vol. 123). Amsterdam: IOS Press.

Cohen, W. W., & Hirsh, H. (1994). Learning the CLASSIC description logic: Theoretical and experimental results. In J. Doyle, E. Sandewall, & P. Torasso (Eds.), *Proceedings of the 4th international conference on principles of knowledge representation and reasoning* (pp. 121–133). San Mateo: Morgan Kaufmann.

Cohen, W. W., Borgida, A., & Hirsh, H. (1993). Computing least common subsumers in description logics. In *Proceedings of the tenth national conference on artificial intelligence* (pp. 754–760). Menlo Park: AAAI Press.

Davies, J., Studer, R., & Warren, P. (Eds.) (2006). *Semantic web technologies—trends and research in ontology-based systems*. New York: Wiley.

[23] See e.g. http://linkeddata.org for further pointers.

de Bruijn, J., Lara, R., Polleres, A., & Fensel, D. (2005). OWL DL vs OWL flight: conceptual modeling and reasoning for the semantic web. In A. Ellis & T. Hagino (Eds.), *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10–14, 2005* (pp. 623–632). New York: ACM.

Dietterich, T., Domingos, P., Getoor, L., Muggleton, S., & Tadepalli, P. (2008). Structured machine learning: the next ten years. *Machine Learning*, *73*(1), 3–23.

Domingos, P. (1998). Occam's two razors: The sharp and the blunt. In *Proceedings of the fourth international conference on knowledge discovery and data mining* (pp. 37–43).

Dutra, I., Page, D., Costa, V. S., & Shavlik, J. (2003). An empirical evaluation of bagging in inductive logic programming. In S. Matwin & C. Sammut (Eds.), *Lecture notes in artificial intelligence: Vol. 2583. Proceedings of the 12th international conference on inductive logic programming* (pp. 48–65). Berlin: Springer.

Esposito, F., Fanizzi, N., Iannone, L., Palmisano, I., & Semeraro, G. (2004). Knowledge-intensive induction of terminologies from metadata. In *The semantic web—ISWC 2004: Third international semantic web conference, Hiroshima, Japan, November 7–11, 2004. Proceedings* (pp. 441–455). Berlin: Springer.

Fanizzi, N., Ferilli, S., Mauro, N. D., & Basile, T. M. A. (2003). Spaces of theories with ideal refinement operators. In G. Gottlob & T. Walsh (Eds.), *IJCAI-03, Proceedings of the eighteenth international joint conference on artificial intelligence, Acapulco, Mexico, August 9–15, 2003* (pp. 527–532). San Mateo: Morgan Kaufmann.

Fanizzi, N., Ferilli, S., Iannone, L., Palmisano, I., & Semeraro, G. (2004). Downward refinement in the ALN description logic. In *HIS* (pp. 68–73). New York: IEEE Computer Society.

Hellmann, S., Lehmann, J., & Auer, S. (2009). Learning of OWL class descriptions on very large knowledge bases. *International Journal on Semantic Web and Information Systems, Special Issue on Scalability and Performance of Semantic Web Systems*, *5*(2), 25–48.

Hitzler, P., Krötzsch, M., & Rudolph, S. (2009). *Foundations of semantic web technologies*. Boca Raton: CRC Press/Chapman & Hall.

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

Horrocks, I., Kutz, O., & Sattler, U. (2006). The even more irresistible SROIQ. In P. Doherty, J. Mylopoulos, & C. A. Welty (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). Menlo Park: AAAI Press.

Iannone, L., & Palmisano, I. (2005). An algorithm based on counterfactuals for concept learning in the semantic web. In *Proceedings of the 18th international conference on industrial and engineering applications of artificial intelligence and expert systems* (pp. 370–379), Bari, Italy.

Iannone, L., Palmisano, I., & Fanizzi, N. (2007). An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, *26*(2), 139–159.

Jiang, N., & Colton, S. (2006). Boosting descriptive ILP for predictive learning in bioinformatics. In S. Muggleton, R. P. Otero, & A. Tamaddoni-Nezhad (Eds.), *Lecture notes in computer science: Vol. 4455. Proceedings of the 15th international conference on inductive logic programming* (pp. 275–289). Berlin: Springer.

Kietz, J.-U., & Morik, K. (1994). A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, *14*, 193–217.

Krötzsch, M., Rudolph, S., & Hitzler, P. (2008a). Description logic rules. In M. Ghallab et al. (Eds.), *Proceedings of the 18th European conf. on artificial intelligence (ECAI-08)* (pp. 80–84). Amsterdam: IOS Press.

Krötzsch, M., Rudolph, S., & Hitzler, P. (2008b). ELP: tractable rules for OWL 2. In A. Sheth et al. (Eds.), *Lecture notes in computer science: Vol. 5318. The semantic web—ISWC 2008, 7th international semantic web conference* (pp. 649–664). Berlin: Springer.

Lehmann, J. (2007). Hybrid learning of ontology classes. In P. Perner (Ed.), *Lecture notes in computer science: Vol. 4571. Machine learning and data mining in pattern recognition, 5th international conference, MLDM 2007, Leipzig, Germany, July 18–20, 2007, Proceedings* (pp. 883–898). Berlin: Springer.

Lehmann, J., & Hitzler, P. (2007a). Foundations of refinement operators for description logics. In H. Blockeel, J. Ramon, J. W. Shavlik, & P. Tadepalli (Eds.), *Lecture notes in computer science: Vol. 4894. Inductive logic programming, 17th international conference, ILP 2007, Corvallis, OR, USA, June 19–21, 2007, Revised selected papers* (pp. 161–174). Berlin: Springer.

Lehmann, J., & Hitzler, P. (2007b). A refinement operator based learning algorithm for the alc description logic. In H. Blockeel, J. Ramon, J. W. Shavlik, & P. Tadepalli (Eds.), *Lecture notes in computer science: Vol. 4894. Inductive logic programming, 17th international conference, ILP 2007, Corvallis, OR, USA, June 19–21, 2007, Revised selected papers* (pp. 147–160). Berlin: Springer.

Lisi, F. A., & Malerba, D. (2003). Ideal refinement of descriptions in AL-log. In T. Horváth (Ed.), *Lecture notes in computer science: Vol. 2835. Inductive logic programming: 13th international conference, ILP 2003, Szeged, Hungary, September 29–October 1, 2003, Proceedings* (pp. 215–232). Berlin: Springer.

Michalski, R. S. (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2*(4), 349–361.

Mitchell, T. (1997). *Machine learning*. New York: McGraw Hill.

Nienhuys-Cheng, S.-H., & de Wolf, R. (Eds.) (1997). *Foundations of inductive logic programming*. *Lecture notes in computer science*, (Vol. 1228). Berlin: Springer.

Nienhuys-Cheng, S.-H., Laer, W. V., Ramon, J., & Raedt, L. D. (1999). Generalizing refinement operators to learn prenex conjunctive normal forms. In S. Džeroski & P. Flach (Eds.), *Lecture notes in artificial intelligence: Vol. 1634. Proceedings of the 9th international workshop on inductive logic programming* (pp. 245–256). Berlin: Springer.

Nienhuys-Cheng, S. H., van der Laag, P. R. J., & van der Torre, L. W. N. (1993). Constructing refinement operators by decomposing logical implication. In P. Torasso (Ed.), *LNAI: Vol. 728. Advances in artificial intelligence: Proceedings of the 3rd congress of the Italian association for artificial intelligence (AI∗IA'93)*, Torino, Italy (pp. 178–189). Berlin: Springer.

Patel-Schneider, P. F., & Horrocks, I. (2007). A comparison of two modelling paradigms in the semantic web. *Journal on Web Semantics*, *5*(4), 240–250.

Rector, A. L., & Brandt, S. (2008). Why do it the hard way? The case for an expressive description logic for SNOMED. *Journal of the American Medical Informatics Association*.

Richards, B. L., & Mooney, R. J. (1995). Refinement of first-order Horn-clause domain theories. *Machine Learning*, *19*(2), 95–131.

Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., & Hart, G. (2008). A comparison of three controlled natural languages for OWL 1.1. In K. Clark & P. F. Patel-Schneider (Eds.), *Proceedings of the fourth international workshop OWL: Experiences and directions, OWLED2008DC*, Washington, DC, April 2008. Available from http://www.webont.org/owled/2008dc.

Shapiro, E. Y. (1991). Inductive inference of theories from facts. In J. L. Lassez & G. D. Plotkin (Eds.), *Computational logic: Essays in honor of Alan Robinson* (pp. 199–255). Cambridge: MIT Press.

Srinivasan, A., King, R. D., Muggleton, S., & Sternberg, M. J. E. (1997). Carcinogenesis predictions using ILP. In S. Džeroski & N. Lavrač (Eds.), *Lecture notes in artificial intelligence: Vol. 1297. Proceedings of the 7th international workshop on inductive logic programming* (pp. 273–287). Berlin: Springer.

Staab, S., & Studer, R. (Eds.) (2004). *Handbook on ontologies*. *International handbooks on information systems*. Heidelberg: Springer.

van der Laag, P. R. J., & Nienhuys-Cheng, S.-H. (1994). Existence and nonexistence of complete refinement operators. In F. Bergadano & L. D. Raedt (Eds.), *Lecture notes in artificial intelligence: Vol. 784. Proceedings of the 7th European conference on machine learning* (pp. 307–322). Berlin: Springer.

Völker, J., Hitzler, P., & Cimiano, P. (2007). Acquisition of OWL DL axioms from lexical resources. In E. Franconi, M. Kifer, & W. May (Eds.), *Lecture notes in computer science: Vol. 4519. Proceedings of the 4th European semantic web conference (ESWC'07)* (pp. 670–685). Berlin: Springer.

Šebelík, J., & Štěpánek, P. (1982). Horn clause programs for recursive functions. In K. Clark & S.-Å. Tärnlund (Eds.), *Logic programming* (pp. 324–340). New York: Academic Press.

Železný, F., Srinivasan, A., & Page, D. (2003). Lattice-search runtime distributions may be heavy-tailed. In S. Matwin & C. Sammut (Eds.), *Lecture notes in artificial intelligence: Vol. 2583. Proceedings of the 12th international conference on inductive logic programming* (pp. 333–345). Berlin: Springer.