A Refinement Operator Based Learning Algorithm for the $\mathcal{ALC}$ Description Logic

Jens Lehmann[1]* and Pascal Hitzler[2]**

[1] Universität Leipzig, Department of Computer Science
Johannisgasse 26, D-04103 Leipzig, Germany,
`lehmann@informatik.uni-leipzig.de`
[2] Universität Karlsruhe (TH), AIFB Institute
D-76128 Karlsruhe, Germany,
`hitzler@aifb.uni-karlsruhe.de`

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, faces a bottleneck due to the lack of available knowledge bases, and it is paramount that suitable automated methods for their acquisition will be developed. In this paper, we provide the first learning algorithm based on refinement operators for the most fundamental description logic $\mathcal{ALC}$. We develop the algorithm from thorough theoretical foundations and report on a prototype implementation.

1 Introduction

The Semantic Web is gaining momentum. Semantic Technologies, based on the same underlying principles, are being applied in adjacent areas such as Software Engineering and Content Management, and industrial interest is rising rapidly. Fundamental to these approaches is the modelling of knowledge by means of ontologies, and the single most popular paradigm for this is by using the Web Ontology Language OWL,[3] which has been recommended by the World Wide Web Consortium (W3C) since 2004.

Progress in research and applications, however, faces a bottleneck due to the lack of available OWL knowledge bases. Considerable effort is therefore currently being invested into developing automated means for the acquisition of ontologies. Most of the currently pursued approaches, however, neglect the expressive power of OWL and are only capable of learning inexpressive ontologies, such as taxonomic hierarchies. As such, they fail by far in leveraging the potential inherent in the expressive features of the Web Ontology Language.

* The first author acknowledges support by the German Federal Ministry of Education and Research (BMBF) under the SoftWiki project (grant 01 ISF02 B).
** The second author acknowledges support by the German Federal Ministry of Education and Research (BMBF) under the SmartWeb project (grant 01 IMD01 B) and by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project.
[3] http://www.w3.org/2004/OWL

From a logical perspective, OWL is basically an expressive description logic (DL) [1]. It is therefore natural to attempt an adaptation of logic-based approaches to machine-learning for automated ontology acquisition. Inspired by the success of Inductive Logic Programming (ILP), we pursue the transfer of ILP methods [13] to DLs, and in this paper we report on a resulting learning algorithm. Our approach is based on a thorough theoretical analysis of the potential and limitations of refinement operators for DLs. We make the following contributions:

1. development of a refinement operator, which conforms to theoretical findings
2. design of an algorithm handling the unavoidable limitations of this operator
3. provision of a preliminary evaluation

The algorithm was created with extensibility to additional (non-$\mathcal{ALC}$) concept constructors, e.g. number restrictions, in mind. In contrast to previous approaches [6,7,8], we pay more attention to finding simple, non-overfitting solutions of the learning problem.

The paper is structured as follows. After some preliminaries in Section 2, we introduce our refinement operator in Section 3 and show that it conforms to the desired theoretical properties. In Section 4 we extend this refinement operator to a learning algorithm. In Section 5, we report on our prototype implementation and preliminary evaluation. We discuss related work in Section 6 and conclude in Section 7. Proofs had to be omitted for lack of space, but can be found in the technical report [10].

2 Preliminaries

2.1 Description Logics

Description logics represent knowledge in terms of *objects*, *concepts*, and *roles*. Objects correspond to constants, concepts to unary predicates, and roles to binary predicates in first order logic. In DL systems information is stored in a *knowledge base*, which is a set of axioms. It is divided in (at least) two parts: *TBox* (terminology) and *ABox* (assertions). The ABox contains *assertions* about objects. It relates objects to concepts and roles. The TBox describes the *terminology* by relating concepts and roles.

We briefly introduce the $\mathcal{ALC}$ description logic, which is the target language of our learning algorithm and refer to [1] for further background on description logics. As usual in logics, interpretations are used to assign a meaning to syntactic constructs. Let N_I denote the set of objects, N_C denote the set of atomic concepts, and N_R denote the set of roles. 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 object $a \in N_I$ an element of $\Delta^{\mathcal{I}}$, to each concept $A \in N_C$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to each role $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Interpretations are extended to concepts as shown in Table 1, and to other elements of a knowledge base in a straightforward way. An interpretation, which satisfies an

construct	syntax	semantics
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
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	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
existential	$\exists r.C$	$(\exists r.C)^{\mathcal{I}} = \{a \mid$ $\exists b.(a,b) \in r^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}\}$
universal	$\forall r.C$	$(\forall r.C)^{\mathcal{I}} = \{a \mid$ $\forall b.(a,b) \in r^{\mathcal{I}}$ implies $b \in C^{\mathcal{I}}\}$

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

axiom (set of axioms) is called a model of this axiom (set of axioms). An $\mathcal{ALC}$ concept is in *negation normal form* if negation only occurs in front of concept names.

If C and D are concepts, then $C \sqsubseteq D$ and $C \equiv D$ are *terminological axioms*. The former axioms are called *inclusions* and the latter *equalities*. An equality whose left hand side is an atomic concept is a *concept definition*.

It is the aim of *inference algorithms* to extract implicit knowledge from a given knowledge base. Standard reasoning tasks include *instance checks*, *retrieval* and *subsumption*. We will only explicitly define the latter. Let C, D be concepts and $\mathcal{T}$ a TBox. *C is subsumed by D*, denoted by $C \sqsubseteq D$, iff for any interpretation $\mathcal{I}$ 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 any model $\mathcal{I}$ 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 $C \equiv D$ ($C \equiv_{\mathcal{T}} D$).

2.2 Learning in Description Logics using Refinement Operators

Definition 1 (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 `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 $\mathcal{K}' \not\models E^-$.*

By Occam's razor [3] simple solutions of the learning problem are to be preferred over more complex ones, because they have a higher predictive quality. We measure simplicity as the *length* of a concept, which is defined in a straightforward way, namely as the sum of the numbers of concept, role, quantifier, and connective symbols occurring in the concept.

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, which is the purpose of *refinement operators*. Intuitively, downward (upward) refinement operators construct specialisations (generalisations) of hypotheses.

A *quasi-ordering* is a reflexive and transitive relation. Let S be a set and $\preceq$ a quasi-ordering on S. In the quasi-ordered space $(S, \preceq)$ a *downward (upward) refinement operator* ρ is a mapping from S to 2^S, such that for any $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$ ($C \preceq C'$). C' is called a *specialisation* (*generalisation*) of C. Quasi-orderings can be used for searching in the space of concepts. As ordering we can use subsumption. If a concept C subsumes a concept D ($D \sqsubseteq C$), then C covers all examples, which are covered by D, which makes subsumption a suitable order.

Definition 2. *A refinement operator in the quasi-ordered space* $(\mathcal{ALC}, \sqsubseteq_{\mathcal{T}})$ *is called an* $\mathcal{ALC}$ refinement operator.

We need to introduce some notions for refinement operators. A *refinement chain of an* $\mathcal{ALC}$ *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. If we look at refinements of an operator ρ, we will often write $C \rightsquigarrow_\rho D$ instead of $D \in \rho(C)$. If the used operator is clear from the context it is usually omitted, i.e. we write $C \rightsquigarrow D$.

We will introduce the concept of weak equality of concepts, which is similar to 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 if for any $C_1 \in S_1$ 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 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 hypothesis.

Definition 3. *An* $\mathcal{ALC}$ *refinement operator* ρ *is called*

- (locally) finite *iff* $\rho(C)$ *is finite for any concept* 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 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 $\mathcal{ALC}$ *downward refinement operator* ρ *is called*

– complete *iff for all concepts C, D with $C \sqsubset_{\mathcal{T}} D$ we can reach a concept E with $E \equiv C$ from D by ρ.*
– weakly complete *iff for all concepts $C \sqsubset_{\mathcal{T}} \top$ we can reach a concept E with $E \equiv C$ from $\top$ by ρ.*

The corresponding notions for upward refinement operators are defined dually.

3 Designing a Refinement Operator

To design a suitable operator, we first look at theoretical limitations. The following theorem from [9] provides a full analysis of the properties of $\mathcal{ALC}$ refinement operators:

Theorem 1 (Property Theorem). *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 $\mathcal{ALC}$ refinement operators (see [9] for details):*

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}*

Incomplete operators are not interesting, because we may then be unable to find possible solutions, so we can ignore the fifth property combination. We can see from the other combinations, that we can have either finity or properness as a property of an $\mathcal{ALC}$ refinement operator – but not both at the same time. Since we are able to handle infinity quite well, as we will describe in Section 4, we will aim for properness. Our learning algorithm will perform a top-down search, so the fourth combination seems to be desirable, because weak completeness is sufficient in this case. However, an incomplete, but weakly complete operator cannot support some of the features which we consider essential in our learning algorithm. Hence, we decided to use the second combination.

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 4 will show how we handle the problems of redundancy and infinity in the learning algorithm.

For each $A \in N_C$, we define $\text{nb}_\downarrow(A) = \{A' \mid A' \in N_C, \text{ there is no } A'' \in N_C \text{ with } A' \sqsubset_{\mathcal{T}} A'' \sqsubset_{\mathcal{T}} A\}$. $\text{nb}_\uparrow(A)$ is defined analogously. In the sequel, we will analyse the refinement operator $\rho_\downarrow$ given by:

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

where the operator $\rho'_\downarrow$ is defined as in Figure 1. The definition refers to a set M which is inductively defined as follows: All elements in $\{A \mid A \in N_C, \text{nb}_\uparrow(A) = \emptyset\}$

(= most general atomic concepts), $\{\neg A \mid A \in N_C, \mathrm{nb}_\downarrow(A) = \emptyset\}$ (= negated most specific atomic concepts), and $\{\exists r.\top \mid r \in N_R\}$ are in M. If a concept C is in M, then $\forall r.C$ with $r \in N_R$ is also in M.

$$\rho'_\downarrow(C) = \begin{cases} \emptyset & \text{if } C = \bot \\ \{C_1 \sqcup \cdots \sqcup C_n \mid C_i \in M \ (1 \leq i \leq n)\} & \text{if } C = \top \\ \{A' \mid A' \in \mathrm{nb}_\downarrow(A)\} \cup \{A \sqcap D \mid D \in \rho'_\downarrow(\top)\} & \text{if } C = A \ (A \in N_C) \\ \{\neg A' \mid A' \in \mathrm{nb}_\uparrow(A)\} \cup \{\neg A \sqcap D \mid D \in \rho'_\downarrow(\top)\} & \text{if } C = \neg A \ (A \in N_C) \\ \{\exists r.E \mid E \in \rho'_\downarrow(D)\} \cup \ \{\exists r.D \sqcap E \mid E \in \rho'_\downarrow(\top)\} & \text{if } C = \exists r.D \\ \{\forall r.E \mid E \in \rho'_\downarrow(D)\} \cup \ \{\forall r.D \sqcap E \mid E \in \rho'_\downarrow(\top)\} & \text{if } C = \forall r.D \\ \quad \cup \{\forall r.\bot \mid D = A \in N_C \text{ and } \mathrm{nb}_\downarrow(A) = \emptyset\} & \\ \{C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \cdots \sqcap C_n \mid & \text{if } C = C_1 \sqcap \cdots \sqcap C_n \\ \quad D \in \rho'_\downarrow(C_i), 1 \leq i \leq n\} & (n \geq 2) \\ \{C_1 \sqcup \cdots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \cdots \sqcup C_n \mid & \text{if } C = C_1 \sqcup \cdots \sqcup C_n \\ \quad D \in \rho'_\downarrow(C_i), 1 \leq i \leq n\} & (n \geq 2) \\ \quad \cup \{(C_1 \sqcup \cdots \sqcup C_n) \sqcap D \mid D \in \rho'_\downarrow(\top)\} & \end{cases}$$

Figure 1. definition of $\rho'_\downarrow$

Proposition 1. *$\rho_\downarrow$ is an $\mathcal{ALC}$ downward refinement operator.*

A distinguishing feature of $\rho_\downarrow$ compared to other refinement operators for learning concepts in DLs [2,6] is that it makes use of the subsumption hierarchy. This is useful, since the operator can make use of knowledge contained implicitly in the TBox. Note that $\rho_\downarrow$ is infinite. The reason is that the set M is infinite and, furthermore, we put no boundary on the number of elements in the disjunctions, which are refinements of the top concept.

3.1 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 4 ($S_\downarrow$)**.** *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 $\forall r.\bot \in S'_\downarrow$, $\exists r.\top \in S'_\downarrow$.*
3. *If $C, C_1, \ldots, C_m$ are in $S'_\downarrow$ then the following concepts are also in $S'_\downarrow$:*
   - *$\exists r.C$, $\quad \forall r.C$, $\quad C_1 \sqcap \cdots \sqcap C_m$, $\quad$ and*
   - *$C_1 \sqcup \cdots \sqcup C_m$ 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 any $\mathcal{ALC}$ concept C there exists a concept $D \in S_\downarrow$ such that $D \equiv C$.

Lemma 1 ($S_\downarrow$). *For any $\mathcal{ALC}$ concept C there exists a concept $D \in S_\downarrow$ such that $D \equiv C$.*

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

Proposition 2 (weak completeness of $\rho_\downarrow$). *$\rho_\downarrow$ is weakly complete.*

Using this, we can prove completeness (again, we refer to [10] for proofs).

Proposition 3. *$\rho_\downarrow$ is complete.*

3.2 Achieving Properness

The operator $\rho_\downarrow$ is not proper, for instance it allows the refinement $\top \rightsquigarrow \exists r.\top \sqcup \forall r.A_1 \quad (\equiv \top)$ where $A_1 \in \text{nb}_\downarrow(\top)$. Indeed, there is no structural subsumption algorithm for $\mathcal{ALC}$ [1], which indicates that it is hard to define a proper operator just by syntactic rewriting rules. One could try to modify $\rho_\downarrow$, such that it becomes proper. Unfortunately, this is likely to lead to incompleteness. Say, we disallow the refinement step just mentioned and consider the following refinement chain:

$$\top \rightsquigarrow \exists r.\top \sqcup \forall r.A_1 \rightsquigarrow \exists r.A_2 \sqcup \forall r.A_1 \quad (A_1, A_2 \in \text{nb}_\downarrow(\top))$$

If we disallow the first step, we would have to ensure that we can reach $\exists r.A_2 \sqcup \forall r.A_1$ from $\top$, otherwise the operator is weakly incomplete. In particular, there can be infinite chains of improper refinements:

$$\top \rightsquigarrow \exists r.\top \sqcup \forall r.A_1 \rightsquigarrow \exists r.(\exists r.\top \sqcup \forall r.A_1) \sqcup \forall r.A_1 \rightsquigarrow \ldots$$

This example illustrates that one would have to allow very complex concepts to be generated as refinements of the top concept, if one wants to achieve weak completeness and properness. So, instead of modifying $\rho_\downarrow$ directly, we allow it to be improper, but consider the closure $\rho_\downarrow^{cl}$ of $\rho_\downarrow$ [2].

Definition 5 ($\rho_\downarrow^{cl}$). *$\rho_\downarrow^{cl}$ is defined as follows: $D \in \rho_\downarrow^{cl}(C)$ iff there exists a refinement chain*

$$C \rightsquigarrow_{\rho_\downarrow} C_1 \rightsquigarrow_{\rho_\downarrow} \ldots \rightsquigarrow_{\rho_\downarrow} C_n = D$$

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

$\rho_\downarrow^{cl}$ is proper by definition. It also inherits the weak completeness of $\rho_\downarrow$, since we do not disallow any refinement steps, but only check whether they are improper. However, it is necessary to show that $\rho_\downarrow^{cl}$ is a meaningful operator, which we will do in the sequel. We already know that $\rho_\downarrow$ is infinite, so it is clear that we cannot consider all refinements of a concept at a time. Therefore, in practise 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 up to a given length are computable in finite time. To show this, we need the following lemma.

Lemma 2 ($\rho_\downarrow$ does not reduce length). *$D \in \rho_\downarrow(C)$ implies $|D| \geq |C|$. Furthermore, there are no infinite refinement chains of the form $C_1 \rightsquigarrow_{\rho_\downarrow} C_2 \rightsquigarrow_{\rho_\downarrow} \dots$ with $|C_1| = |C_2| = \dots$, i.e. after a finite number of steps we reach a strictly longer concept.*

Proposition 4 (usefulness of $\rho^{cl}_\downarrow$). *For any concept C in negation normal form and any natural number n, the set $\{D \mid D \in \rho^{cl}_\downarrow(C), |D| \leq n\}$ can be computed in finite time.*

Due to Proposition 4 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 $O(|N_C| \cdot |C|)$ steps a refinement of greater length will be reached, which bounds the complexity of computing the closure.

4 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.

4.1 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.[4] We can still reach any concept, 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 as efficiently as possible.

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

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 $A_1 \sqcap A_2 \simeq A_2 \sqcap A_1$. In conjunctions and disjunctions, we have 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 $\mathcal{ALC}$ 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 $\mathcal{ALC}$ 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 (PSPACE 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.

4.2 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 heuristics decides which nodes to expand. To define a search heuristics for our learning algorithm, we need some notions to be able to express what we consider a good concept.

Definition 6 (quality). *Let $\mathcal{K}$ be a knowledge base, E^- the set of negative examples, and E^+ the set of positive examples of a learning problem. The* quality *of a concept C is a function, which maps a concept to an element of $\mathcal{Q}$ with $\mathcal{Q} = \{0, \dots, -|E^-|\} \cup \{tw\}$, defined by $q(C) = tw$ if there is an $e \in E^+$ with $\mathcal{K} \cup \{C\} \not\models e$ and $q(C) = -|\{e \mid e \in E^- and\ \mathcal{K} \cup \{C\} \models e\}|$ otherwise.*

The quality of a concept is "tw" if it is too weak, i.e. it does not cover all positive examples. In all other cases, we assign a number n with $n \geq 0$ to a concept, which is the number of negative examples covered.

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. 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 quadruple (C, n, q, b), where C is an $\mathcal{ALC}$ concept, $n \in \mathbb{N}$ is the *horizontal expansion*, $q \in \mathcal{Q} \cup \{\text{-}\}$ is the *quality* (- stands for non-evaluated quality), and $b \in \{\text{true}, \text{false}\}$ is a boolean marker for the redundancy of a node.

The search heuristics selects the fittest node in the search tree at a given time. We define fitness as a lexicographical order over quality and horizontal expansion.

Definition 7 (fitness). *Let $N_1 = (C_1, n_1, q_1, b_1)$ and $N_2 = (C_2, n_2, q_2, b_2)$ be two nodes with defined quality ($q_1, q_2 \neq -, tw$). N_1 is* fitter *than N_2, denoted by $N_2 \leq_f N_1$ iff $q_2 < q_1$ or $q_1 = q_2$ and $n_1 \leq n_2$.*

Note, that we use horizontal expansion instead of concept length as second criterion, which makes the algorithm more flexible in searching less explored areas of the search space. More sophisticated ways of ordering concepts are also possible, e.g. tradeoffs between quality and horizontal expansion. Such fitness heuristics enable the algorithm to handle noise. The fitness function can be defined independently of the core learing 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.

Algorithm 1: learning algorithm

```
   Input: horizExpFactor in ]0,1]
1  ST (search tree) is set to the tree consisting only of the root node
   $(\top, 0, q(\top), false)$
2  $minHorizExp = 0$
3  while ST does not contain a correct concept do
4      choose $N = (C, n, q, b)$ with highest fitness in ST
5      expand $N$ up to length $n + 1$, i.e. :
6      begin
7          add all nodes $(D, n, -, checkRed(ST, D))$ with $D \in transform(\rho_{\downarrow}^{cl}(C))$
           and $|D| = n + 1$ as children of $N$
8          evaluate created non-redundant nodes
9          change $N$ to $(C, n + 1, q, b)$
10     end
11     $minHorizExp = \max(minHorizExp, \lceil horizExpFactor * (n + 1)\rceil)$
12     while there are nodes with defined quality and horiz. expansion smaller
       minHorizExp do
13         expand these nodes up to minHorizExp
14 Return a correct concept in ST
```

We see, that the usual expansion in a search algorithm is replaced by a one step horizontal expansion. If we only expand the fittest node, we may not explore large parts of the search space. In order to avoid this, a minimum horizontal expansion factor is used, which specifies that all nodes have to be expanded at least up to this length. This factor allows us to control the tradeoff between expanding only the fittest nodes and exploring other parts of the search space.

Correctness of the algorithm can be shown:

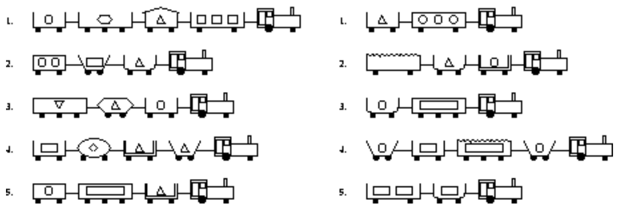

Figure 2. Michalski trains

Proposition 5 (correctness). *If a learning problem has a solution, then Algorithm 1 terminates and computes a correct solution of the learning problem.*

5 Preliminary Evaluation

We want to illustrate our algorithm using Michalski's trains [12] as an 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 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 `Train`, which cover all examples. Other atomic concepts are too weak to be considered for further exploration. The exploration of the top concept leads to $\exists$`hasCar`.$\top$, which is then expanded to $\exists$`hasCar.Closed`. This covers all positives and two negatives. The heuristic picks this node and extends it 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 arches [14], learning poker hands, and understanding the moral reasoning of humans. The two latter examples were taken from the UCI Machine Learning repository[5]. 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.

The arch problem is small in terms of size and complexity of the background knowledge. The poker example is larger in terms of size, but still not very complex. The moral reasoner, however, is an expressive ontology, which we derived from a theory given as logic program. The solutions of the examples cover a range of different concept constructors and are of varying length and complexity.

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

For all test runs we used a (non-optimised) horizontal expansion factor of 0.6. As a reasoner we used Pellet[6] (version 1.4RC1), which was connected to the learning program using the DIG 1.1 interface[7] on a 1.4 GHz CPU machine. The only system we could use for comparison is YinYang [8]. The system in [5] is no longer available and the approach in [2] was not fully implemented.

Table 2 summarises the results we obtained. In all cases, our implementation – called DL-Learner – was able to learn the shortest correct definition (which coincides with the intended solution of these problems). YinYang produces longer solutions and could not solve the second poker problem (it produces an error after trying to compute most specific concepts for some time). It generated an incorrect answer for both moral reasoner problems. The percentage of time spent for reasoner requests increases with the complexity and size of background knowledge and the number of examples (it is $> 99\%$ for the moral reasoner problems), which shows that minimizing the number of reasoner requests, e.g. by redundancy elimination, is an important issue.

problem	axioms, concepts, roles	DL-Learner			YinYang		
	objects, examples	runtime	length	correct	runtime	length	correct
trains	252, 8, 5, 50, 10	1.1s	5	100%	2.3s	8	100%
arches	71, 6, 5, 19, 5	4.6s	9	100%	1.5s	23	100%
moral (simple)	2176, 43, 4, 45, 43	17.7s	3	100%	205.3s	69	67.4%
moral (complex)	2107, 40, 4, 45, 43	88.1s	8	100%	181.4s	70	69.8%
poker (pair)	1335, 2, 6, 311, 49	7.7s	5	100%	17.1s	43	100%
poker (straight)	1419, 2, 6, 347, 55	35.6s	11	100%	-	-	-

Table 2. evaluation results for DL-Learner and YinYang

6 Related Work

An interesting paper close to our work is [2]. It suggests a refinement operator for the $\mathcal{ALER}$ description logic. They also investigate some theoretical properties of refinement operators. As we have done with the design of $\rho_\downarrow$, they favour the use of a downward refinement operator to enable a top-down search. The authors use $\mathcal{ALER}$ *normal form* (see the paper for a detailed description), 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 easier to such extensions. We also deal quite differently with infinity, show how the subsumption hierarchy of atomic concepts can be used, and describe how redundancy can be avoided efficiently.

[6] http://pellet.owldl.com
[7] http://dl.kr.org/dig/

A second area of ongoing related work is described in [6,7,8]. They take a different approach 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 [4,5] in learning in DLs has mostly focused on approaches using least common subsumers, which face this problem to an even larger extent.

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 solutions. The produced solutions will be close to the shortest solution in negation normal form and, thus, overfitting is unlikely.

Another related area of research are approaches for learning in the hybrid language AL-log [11], which combines $\mathcal{ALC}$ with the function free Horn clause language Datalog.

7 Conclusions and Further Work

To the best of our knowledge, our work presents the first refinement operator based learning algorithm for expressive DLs which are closed under boolean operations.[8] It is based on thorough theoretical investigations concerning the potential of using refinement operators for DLs, 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 satisfiable manner, allowing us to specify a learning algorithm which we proved to be correct. We implemented the algorithm and an evaluation showed the feasibility of our approach.

Future work will focus on increasing the expressiveness of the learned language, integrating the learning algorithm in an ontology editor, creating benchmark datasets, and testing on real world data sets.

References

1. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
2. Liviu Badea and Shan-Hwei Nienhuys-Cheng. A refinement operator for description logics. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 40–59. Springer-Verlag, 2000.
3. Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*, pages 201–204. Morgan Kaufmann, 1990.

[8] To be precise, [8] also uses refinement operators, but not as centrally as in our approach – see Section 6.

4. William W. Cohen, Alex Borgida, and Haym Hirsh. Computing least common subsumers in description logics. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 754–760. AAAI Press, 1993.
5. William W. Cohen and Haym Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 121–133. Morgan Kaufmann, may 1994.
6. Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. 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*, pages 441–455. Springer, 2004.
7. Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Concept formation in expressive description logics. In *Machine Learning: ECML 2004, 15th European Conference on Machine Learning, Pisa, Italy, September 20-24, 2004, Proceedings*. Springer, 2004.
8. Luigi Iannone and Ignazio Palmisano. 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*, pages 370–379, Bari, Italy, June 2005.
9. Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*, 2007.
10. Jens Lehmann and Pascal Hitzler. A refinement operator based learning algorithm for the $\mathcal{ALC}$ description logic. Technical report, University of Leipzig, 2007. Downloadable from http://www.jens-lehmann.org.
11. Francesca A. Lisi and Donato Malerba. Ideal refinement of descriptions in AL-log. In Tamás Horváth, editor, *Inductive Logic Programming: 13th International Conference, ILP 2003, Szeged, Hungary, September 29-October 1, 2003, Proceedings*, volume 2835 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2003.
12. R. S. Michalski. Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4):349–361, 1980.
13. Shan-Hwei Nienhuys-Cheng and Ronald de Wolf, editors. *Foundations of Inductive Logic Programming*. Lecture Notes in Computer Science. Springer, 1997.
14. P. Winston. Learning structural descriptions from examples. In P. Winston, editor, *The Psychology of Computer Vision*, pages 157–209. McGraw-Hill, New York, 1975.