Ideal Downward Refinement in the $\mathcal{EL}$ Description Logic

Jens Lehmann[1] and Christoph Haase[2]

[1] Universität Leipzig, Department of Computer Science,
Johannisgasse 26, D-04103 Leipzig, Germany,
lehmann@informatik.uni-leipzig.de
[2] Oxford University Computing Laboratory,
Wolfson Building, Parks Rd, Oxford, OX1 3QD, United Kingdom,
christoph.haase@comlab.ox.ac.uk

Abstract. With the proliferation of the Semantic Web, there has been a rapidly rising interest in description logics, which form the logical foundation of the W3C standard ontology language OWL. While the number of OWL knowledge bases grows, there is an increasing demand for tools assisting knowledge engineers in building up and maintaining their structure. For this purpose, concept learning algorithms based on refinement operators have been investigated. In this paper, we provide an ideal refinement operator for the description logic $\mathcal{EL}$ and show that it is computationally feasible on large knowledge bases.

1 Introduction

The Semantic Web is steadily growing[3] 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. 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 is a key to the growth of the Semantic Web.

However, 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. [5]). In particular, concept learning methods have attracted interest, see e.g. [2,6,11,13].

[3] As a rough size estimate, the semantic index Sindice (http://sindice.com/) lists more than 10 billion entities from more than 100 million web pages.

Many concept learning methods borrow ideas from Inductive Logic Programming including the use of *refinement operators*. Properties like ideality, completeness, finiteness, properness, minimality, and non-redundancy are used as theoretical criteria for the suitability of such operators. It has been shown in [12] that no ideal refinement operator for DLs such as $\mathcal{ALC}$, $\mathcal{SHOIN}$, and $\mathcal{SROIQ}$ can exist (the two latter DLs are underlying OWL and OWL 2, respectively). In this article, an important gap in the the analysis of refinement operator properties is closed by showing that ideal refinement operators for the DL $\mathcal{EL}$ do exist, which in turn can lead to an advance in DL concept learning.

$\mathcal{EL}$ is a light-weight DL, but despite its limited expressive power it has proven to be of practical use in many real-world large-scale applications. For example, the Systematized Nomenclature of Medicine Clinical Terms (Snomed CT) [4] and the Gene Ontology [18] are based on $\mathcal{EL}$. Since standard reasoning in $\mathcal{EL}$ is polynomial, it is suitable for large ontologies. It should furthermore be mentioned that $\mathcal{EL}^{++}$, an extension of $\mathcal{EL}$, will become one of three profiles in the upcoming standard ontology language OWL 2.

Overall, we make the following contributions in this paper: We

- close a gap in the research of properties of refinement operators in DLs,
- provide an ideal and practically useful refinement operator for $\mathcal{EL}$, and
- show the computational feasibility of the operator.

This paper is structured as follows. Section 2 introduces the preliminaries for our work and the refinement operator is presented in Section 3. There, we prove its ideality and describe how it can be optimised to work efficiently and incorporate background knowledge. We evaluate the operator on real-world knowledge bases in Section 4. Related work is described in Section 5 and conclusions are drawn in Section 6.

2 Preliminaries

In this section, the definitions relevant for the refinement operator in Section 3 are introduced. Besides recalling known facts from the literature, we introduce minimal $\mathcal{EL}$ trees that serve as the basis for the refinement operator.

2.1 The $\mathcal{EL}$ Description Logic

Before we begin to introduce the DL $\mathcal{EL}$, we briefly recall some notions from order theory. Let Q be a set and $\preceq$ a quasi order on Q, i.e., a reflexive and transitive binary relation on Q. Then $(Q, \preceq)$ is called a *quasi ordered space*. The quasi order $\preceq$ induces the equivalence relation $\simeq$ and the *strict quasi order* $\prec$ on Q: $q \simeq q'$ iff $q \preceq q'$ and $q' \preceq q$, and $q \prec q'$ iff $q \preceq q'$ and $q \not\simeq q'$. For $P \subseteq Q$, $max(P) := \{p \in P \mid \text{there is no } p' \in P \text{ with } p \prec p'\}$ defines the *set of maximal elements* of P. We say $(Q, \preceq)$ has a *greatest element* iff there is a $q^* \in Q$ such that $max(Q) := \{q^*\}$.

Concept constructor	Syntax	Semantics
Top	$\top$	$\Delta^{\mathcal{I}}$
Concept name	A	$A^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \text{there is } y \in C^{\mathcal{I}} \text{ with } (x, y) \in r^{\mathcal{I}}\}$

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

Name	Syntax	Restriction on $\mathcal{I}$
Concept inclusion	$A \sqsubseteq B$	$A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$
Role inclusion	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
Disjointness	$A \sqcap B \equiv \bot$	$A^{\mathcal{I}} \cap B^{\mathcal{I}} = \emptyset$
Domain	$domain(r) = A$	$x \in A^{\mathcal{I}}$ for all $(x, y) \in r^{\mathcal{I}}$
Range	$range(r) = A$	$y \in A^{\mathcal{I}}$ for all $(x, y) \in r^{\mathcal{I}}$

Table 2. Knowledge base axioms.

The expressions in the DL $\mathcal{EL}$ are *concepts*, which are built inductively starting from sets of *concepts names* N_C and *role names* N_R of arbitrary but finite cardinality, and then applying the *concept constructors* shown in Table 1. There and in the following, A, B denote concept names, r, s denote role names, and C, D denote arbitrary $\mathcal{EL}$ concepts. By $\mathcal{C}(\mathcal{EL})$ we refer to the set of all $\mathcal{EL}$ concepts. The size of an $\mathcal{EL}$ concept C is denoted by $|C|$ and is is just the number of symbols used to write it down. When proving properties of $\mathcal{EL}$ concepts, the *role depth* of a concept C is a useful induction argument. It is defined by structural induction as $rdepth(A) = rdepth(\top) := 0$, $rdepth(C \sqcap D) := max(rdepth(C), rdepth(D))$ and $rdepth(\exists r.C) := rdepth(C) + 1$.

The semantics of an $\mathcal{EL}$ concept C is given in terms of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a set called the *interpretation domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function*. The interpretation function maps each $A \in N_C$ to a subset of $\Delta^{\mathcal{I}}$, and each $r \in N_R$ to a binary relation on $\Delta^{\mathcal{I}}$. It is then inductively extended to arbitrary $\mathcal{EL}$ concepts as shown in Table 1.

In this paper, a *knowledge base* $\mathcal{K}$ is a finite union of knowledge base axioms given in Table 2. An interpretation $\mathcal{I}$ is a *model* of a knowledge base $\mathcal{K}$ iff the conditions on the right-hand side of Table 2 are fulfilled for every knowledge base axiom in $\mathcal{K}$. An $\mathcal{EL}$ concept C is *satisfiable* w.r.t. $\mathcal{K}$ iff there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$.

A standard reasoning task in DLs is *subsumption*. Given a knowledge base $\mathcal{K}$ and $\mathcal{EL}$ concepts C, D, we say C is *subsumed* by D w.r.t. $\mathcal{K}$ ($C \sqsubseteq_{\mathcal{K}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{K}$. Intuitively, this states that the concept C is a specialisation of the concept D w.r.t. $\mathcal{K}$. In the remainder of this paper, we always assume a knowledge base to be implicitly present, and we therefore just write $C \sqsubseteq D$. Obviously, $(\mathcal{C}(\mathcal{EL}), \sqsubseteq)$ forms a quasi ordered space, from which we can accordingly derive the relations $\equiv$ (*equivalence*) and $\sqsubset$ (*strict subsumption*).

Example 1. From the following knowledge base $\mathcal{K}$ we can infer $C \sqsubseteq D$.

$$\begin{aligned}
N_C &= \{\texttt{Human}, \texttt{Animal}, \texttt{Bird}, \texttt{Cat}\} \\
N_R &= \{\texttt{has}, \texttt{has\_child}, \texttt{has\_pet}\} \\
\mathcal{K} &= \{\texttt{has\_pet} \sqsubseteq \texttt{has}, \texttt{has\_child} \sqsubseteq \texttt{has}, \texttt{Bird} \sqsubseteq \texttt{Animal}, \texttt{Cat} \sqsubseteq \texttt{Animal}, \\
&\qquad \mathit{domain}(\texttt{has\_pet}) = \texttt{Animal}\} \\
C &= \texttt{Human} \sqcap \exists \texttt{has\_pet}.\top \sqcap \exists.\texttt{has\_child}.\top \\
D &= \texttt{Human} \sqcap \exists \texttt{has}.\texttt{Animal}
\end{aligned}$$

In practice, knowledge bases can be derived from arbitrary ontologies, which may be formulated in DLs other than $\mathcal{EL}$. Concept and role inclusion axioms can be extracted by computing a classification of the respective ontology, and the remaining axioms can be handled in a similar fashion. The presented $\mathcal{EL}$ refinement operator can therefore be used with all DL knowledge bases.

For a role name $r \in N_R$, we define the set of role names that are strictly below r in the subsumption hierarchy as $sh_{\downarrow}(r) := max\{s \mid s \sqsubset r\}$. Given finite sets of concept names $\mathcal{A}, \mathcal{B} \subseteq N_C$, we write $\mathcal{A} \sqsubseteq \mathcal{B}$ iff for every $B \in \mathcal{B}$ there is some $A \in \mathcal{A}$ such that $A \sqsubseteq B$. We sometimes abuse notation and write $\mathcal{A} \sqsubseteq B$ instead of $\mathcal{A} \sqsubseteq \{B\}$. We call $\mathcal{A} \subseteq N_C$ *reduced* if there does not exist $\mathcal{B} \subseteq N_C$ with $|\mathcal{B}| < |\mathcal{A}|$ and $\mathcal{A} \equiv \mathcal{B}$.

2.2 Downward Refinement Operators

Refinement operators are used to structure a search process for concepts. Intuitively, downward refinement operators construct specialisations of hypotheses. This idea is well-known in Inductive Logic Programming [15].

Let $(Q, \preceq)$ be a quasi ordered space and denote by $\mathcal{P}(Q)$ the powerset of Q. A mapping $\rho : Q \rightarrow \mathcal{P}(Q)$ is a *downward refinement operator* on $(Q, \preceq)$ iff $q' \in \rho(q)$ implies $q' \preceq q$. In the remainder of this paper, we will call downward refinement operators just refinement operators. We write $q \leadsto_\rho q'$ for $q' \in \rho(q)$ and drop the index ρ if the refinement operator is clear from the context. A *refinement chain of length* n of a refinement operator ρ that starts in q_1 and ends in q_n is a sequence $q_1 \leadsto \ldots \leadsto q_n$ such that $q_i \leadsto q_{i+1}$ for $1 \leq i < n$. We say that the chain *goes through* q iff $q \in \{q_1, \ldots, q_n\}$. Moreover, $q \leadsto^* q'$ iff there exists a refinement chain of length n starting from q and ending in q' for some $n \in \mathbb{N}$.

Refinement operators can be classified by means of their properties. Let $(Q, \preceq)$ be a quasi ordered space with a greatest element, and let $q, q', q'' \in Q$. A refinement operator ρ is *finite* iff $\rho(q)$ is finite for all q. It is *proper* iff $q \leadsto q'$ implies $q \not\equiv q'$. We call ρ *complete* iff $q' \prec q$ implies $q \leadsto^* q''$ for some $q'' \equiv q'$. Let q^* be the greatest element in $(Q, \preceq)$, ρ is *weakly complete* iff for any $q' \prec q^*$, $q^* \leadsto^* q''$ with $q'' \equiv q'$. We say ρ is *redundant* iff $q^* \leadsto^* q'$ via two refinement chains, where one goes through an element q'' and the other one does not go through q''. Finally, ρ is *ideal* iff it is finite, proper and complete.

2.3 Minimal $\mathcal{EL}$ Concepts

An important observation is that $\mathcal{EL}$ concepts can be viewed as directed labeled trees, see e.g. [1]. This allows for deciding subsumption between concepts in terms of the existence of a simulation relation between the nodes of their corresponding trees. Moreover, the graph approach to $\mathcal{EL}$ concepts allows for a canonical representation of $\mathcal{EL}$ concepts as minimal $\mathcal{EL}$ trees. The latter generalise similar approaches found in the literature, namely "reduced $\mathcal{EL}$ concept terms" [10] and "minimal XPath tree pattern queries" [16]. Most proofs are omitted in this section and deferred to the full version of this paper, since they are mostly a straight-forward generalisation of the proofs found in [10].

An $\mathcal{EL}$ *graph* is a directed labeled graph $G = (V, E, \ell)$, where V is the finite set of *nodes*, $E \subseteq V \times N_R \times V$ is the set of *edges*, and $\ell : V \to \mathcal{P}(N_C)$ is the *labeling function*. We define $V(G) := V$, $E(G) := E$, $\ell(G) := \ell$ and $|G| := |V| + |E|$. For an edge $(v, r, w) \in E$, we call w an $(r\text{-})$*successor* of v, and v an $(r\text{-})$*predecessor* of w. Given a node $v \in V$, a labelling function ℓ and $L \subseteq N_C$, we define $\ell[v \mapsto L]$ as $\ell[v \mapsto L](v) := L$ and $\ell[v \mapsto L](w) := \ell(w)$ for all $w \neq v$. Given G and $v \in V(G)$, we define $G[v \mapsto L] := (V(G), E(G), \ell(G)[v \mapsto L])$. We say $v_1 \xrightarrow{r_1} \cdots \xrightarrow{r_n} v_{n+1}$ is a *path* of length n from v_1 to v_{n+1} in G iff $(v_i, r_i, v_{i+1}) \in E$ for $1 \leq i \leq n$. A graph G contains a cycle iff there is a path $v \xrightarrow{r_1} \cdots \xrightarrow{r_n} v$ in G.

An $\mathcal{EL}$ concept is represented by an $\mathcal{EL}$ *concept tree*, which is a connected finite $\mathcal{EL}$ graph t that does not contain any cycle, has a distinguished node called the *root* of t that has no predecessor, and every other node has exactly one predecessor along exactly one edge. The set of $\mathcal{EL}$ concept trees is denoted by T. In the following, we call an $\mathcal{EL}$ concept tree just a tree. Figure 1 illustrates two examples of such trees. Given a tree t, we denote by $\mathit{root}(t)$ its root. The tree t corresponding to a concept C is defined by induction on $n = \mathit{rdepth}(C)$. For $n = 0$, t consists of a single node that is labelled with all concepts names occurring in C. For $n > 0$, the root of t is labelled with all concept names occurring on the top-level of C. Furthermore, for each existential restriction $\exists r.D$ on the top-level of C, it has an r-labelled edge to the root of a subtree of t' which corresponds to D. As an example, the tree t corresponding to $A_1 \sqcap \exists r.A_2$ is $t = (\{v_1, v_2\}, \{(v_1, r, v_2)\}, \ell)$ where ℓ maps v_1 to $\{A_1\}$ and v_2 to $\{A_2\}$. By $t_\top$ we denote the tree corresponding to $\top$. Obviously, the transformation from a concept to a tree can be performed in linear time w.r.t. the size of the concept. Similarly, any tree has a corresponding concept[4], and the transformation can be performed in linear time, too.

Let t, t' be trees, $v \in V(t)$ and assume w.l.o.g. that $V(t) \cap V(t') = \emptyset$. Denote by $t[v \leftarrow (r, t')]$ the tree obtained from plugging t' via an r-edge into the node v of t, i.e. the tree $(V(t) \cup V(t'), E(t) \cup E(t') \cup \{(v, r, \mathit{root}(t'))\}, \ell \cup \ell')$, where $\ell \cup \ell'$ is the obvious join of the labeling functions of t and t'. By $t(v)$ we denote the subtree at v. Let C be a concept and t the tree corresponding to C. We define $\mathit{depth}(t) := \mathit{rdepth}(C)$, and for $v \in V(t)$, $\mathit{level}(v) := \mathit{depth}(t) - \mathit{depth}(t(v))$.

[4] Strictly speaking, t has a set of corresponding concepts, which are all equivalent up to commutativity.

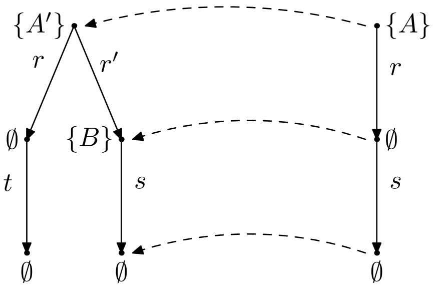

Fig. 1. A (non-maximal) simulation relation w.r.t. the knowledge base $\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$ from the tree corresponding to $A \sqcap \exists r.\exists s.\top$ to the tree corresponding to $A' \sqcap \exists r.\exists t.\top \sqcap \exists r'.(B \sqcap \exists s.\top)$.

Moreover, $\mathit{onlevel}(t, n)$ is the set of nodes $\{v \mid \mathit{level}(v) = n\}$ that appear on level n.

Definition 1. *Let $t = (V, E, \ell), t' = (V', E', \ell')$ be trees. A* simulation relation *from t' to t is a binary relation $\mathcal{S} \subseteq V \times V'$ such that if $(v, v') \in \mathcal{S}$ then the following* simulation conditions *are fulfilled:*

(SC1) $\ell(v) \sqsubseteq \ell'(v')$
(SC2) for every $(v', r, w') \in E'$ there is $(v, r, w) \in E_1$ such that $r \sqsubseteq r'$ and $(w, w') \in \mathcal{S}$

We write $t \preceq t'$ if there exists a simulation relation $\mathcal{S}$ from t' to t such that $(\mathit{root}(t), \mathit{root}(t')) \in \mathcal{S}$. It is easily checked that $(T, \preceq)$ forms a quasi ordered space, and we derive the relations $\simeq$ and $\prec$ accordingly. A simulation $\mathcal{S}$ from t' to t is *maximal* if for every simulation $\mathcal{S}'$ from t' to t, $\mathcal{S}' \subseteq \mathcal{S}$. It is not hard to check that $\mathcal{S}$ is unique. Using a dynamic programming approach, the maximal simulation can be computed in $\mathcal{O}(|t| \cdot |t'|)$. Figure 1 shows an example of a simulation.

The following lemma is proven by induction on $\mathit{rdepth}(D)$. It allows us to decide subsumption between concepts C, D in terms of the existence of a simulation between their corresponding trees t, t', and moreover to interchange concepts and their corresponding trees. For that reason, the $\mathcal{EL}$ refinement operator presented in the next section will work on trees rather than concepts.

Lemma 1. *Let C, D be concept with their corresponding trees t, t'. Then $C \sqsubseteq D$ iff $t \preceq t'$.*

We can now introduce minimal $\mathcal{EL}$ trees which serve as a canonical representation of equivalent $\mathcal{EL}$ concepts.

Definition 2. *Let $t = (V, E, \ell)$ be a tree. We call t* label reduced *if for all $v \in V$, $\ell(v)$ is reduced, i.e. no concept name can be removed from the label without resulting in an inequivalent tree. Moreover, t* contains redundant subtrees *if there are $(v, r, w), (v, r', w') \in E$ with $w \neq w'$, $r \sqsubseteq r'$ and $t(w) \preceq t(w')$. We call t* minimal *if t is label reduced and does not contain redundant subtrees.*

It follows that the minimality of a tree t can be checked in $\mathcal{O}(|t|^2)$ by computing the maximal simulation from t to t and then checking for each $v \in V(t)$ whether v is label reduced and, using $\mathcal{S}$, whether v is not the root of redundant subtrees. The set of minimal $\mathcal{EL}$ trees is denoted by T_{min}.

We close this section with a small lemma that will be helpful in the next section.

Lemma 2. *Let T_n be the set of minimal $\mathcal{EL}$ trees up to depth $n \geq 0$, and let t, t' be $\mathcal{EL}$ trees with $depth(t) < depth(t')$. Then the following holds:*

1. *$|T_n|$ is finite*
2. *$t \not\lesssim t'$*

3 An Ideal $\mathcal{EL}$ Refinement Operator

In this section, we define an ideal refinement operator. In the first part, we are more concerned with a description of the operator on an abstract level, which allows us to prove its properties. The next part addresses optimisations of the operator that improve its performance in practice.

3.1 Definition of the Operator

For simplicity, we subsequently assume the knowledge base to only contain concept and role inclusion axioms. We will sketch in the next section how the remaining restriction axioms can be incorporated in the refinement operator.

The refinement operator ρ, to be defined below, is a function that maps a tree $t \in T_{min}$ to a subset of T_{min}. It can be divided into the three base operations *label extension*, *label refinement* and *edge refinement*. Building up on that, the complex operation *attach subtree* is defined. Each such operation takes a tree $t \in T_{min}$ and a node $v \in V(t)$ as input and returns a set of trees that are refined at node v. Figure 2 provides an example.

The base operations are as follows: the operation $e\ell(t, v)$ returns the set of those minimal trees that are derived from t by extending the label of v. Likewise, $r\ell(t, v)$ is the set of minimal trees obtained from t by refining the label of v. Last, $re(t, v)$ is obtained from t by refining one of the outgoing edges at v. Formally,

- $e\ell(t, v)$: $t' \in e\ell(t, v)$ iff $t' \in T_{min}$ and $t' = t[v \mapsto (\ell(v) \cup \{A\})]$, where $A \in max\{B \in N_C \mid \ell(v) \not\sqsubseteq B\}$

- $r\ell(t, v)$: $t \in r\ell(t, v)$ iff $t' \in T_{min}$ and $t' = t[v \mapsto (\ell(v) \cup \{A\}) \setminus \{B\})]$, where $B \in \ell(v)$, $A \in max\{A' \in N_C \mid A' \sqsubset B\}$ and there is no $B' \in \ell(v)$ with $B \neq B'$ and $A \sqsubset B$

- $re(t, v)$: $t' \in re(t, v)$ iff $t' \in T_{min}$ and $t' = (V, E', \ell)$, where $E' = E \setminus \{(v, r, w)\} \cup \{(v, r', w)\}$ for some $(v, r, w) \in E$ and $r' \in sh_{\downarrow}(r)$

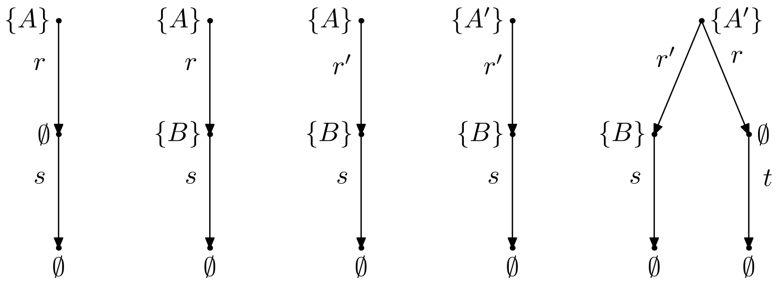

Fig. 2. The tree on the left is refined stepwise to the tree on the right, where we assume a knowledge base $\mathcal{K} = \{A' \sqsubseteq A, r' \sqsubseteq r\}$. The operator performs four different kinds of operations (from left to right): 1. label extension (B added), 2. edge refinement (r replaced by r'), 3. label refinement (A replaced by A'), 4. attaching a subtree ($\exists r.\exists t.\top$ added).

The crucial part of the refinement operator is the attach subtree operation, which is defined by Algorithm 1. The set $as(t, v)$ consists of minimal trees obtained from t that have an extra subtree attached to v. It recursively calls the refinement operator ρ and we therefore give its definition before we explain $as(t, v)$ in more detail.

Definition 3. *The refinement operator $\rho : T_{min} \to \mathcal{P}(T_{min})$ is defined as:*

$$\rho(t) := \bigcup_{v \in V(t)} (e\ell(t, v) \cup r\ell(t, v) \cup re(t, v) \cup as(t, v))$$

For $t \in T_{min}$ and $v \in V$, Algorithm 1 keeps a set of output trees $\mathcal{T}$ and a set $\mathcal{M}$ of candidates which are tuples consisting of a minimal $\mathcal{EL}$ tree and a set of role names. Within the first while loop, an element $(t', \mathcal{R})$ is removed from $\mathcal{M}$. The set $\mathcal{R}'$ is initialized to contain the greatest elements of $\mathcal{R}$, and $\mathcal{R}''$ is initially empty and will later contain role names that need further inspection. In the second while loop, the algorithm iterates over all role names r in $\mathcal{R}'$. First, the tree t'' is constructed from t by attaching the subtree (v, r, w) to v, where w is the root of t'. It is then checked whether t'' is minimal. If this is the case, t'' is a refinement of t and is added to $\mathcal{T}$. Otherwise there are two reasons why t'' is not minimal: Either the newly attached subtree is subsumed by some other subtree of t, or the newly attached subtree subsumes some other subtree of t. The latter case is checked in Line 11, and if it applies the algorithm skips the loop. This prevents the algorithm from running into an infinite loop, since we would not be able to refine t' until t'' becomes a minimal tree. Otherwise in the former case, we proceed in two directions. First, $sh_{\downarrow}(r)$ is added to $\mathcal{R}'$, so it can be checked in the next round of the second while loop whether t' attached via some $r' \in sh_{\downarrow}(r) \cap \mathcal{R}$ to v yields a refinement. Second, we add r to $\mathcal{R}''$, which can be seen as "remembering" that r did not yield a refinement in connection

Algorithm 1 Computation of the set $as(t, v)$

```
 1: $\mathcal{T} := \emptyset$; $\mathcal{M} := \{(t_\top, N_R)\}$;
 2: while $\mathcal{M} \neq \emptyset$ do
 3:   choose and remove $(t', \mathcal{R}) \in \mathcal{M}$;
 4:   $\mathcal{R}' := max(\mathcal{R})$; $\mathcal{R}'' := \emptyset$;
 5:   while $\mathcal{R}' \neq \emptyset$ do
 6:     choose and remove $r \in \mathcal{R}'$;
 7:     $t'' := t[v \leftarrow (r, t')]$; $w := root(t')$;
 8:     if $t''$ is minimal then
 9:       $\mathcal{T} := \mathcal{T} \cup \{t''\}$;
10:     else
11:       for all $(v, r', w') \in E(t'')$ with $w \neq w'$ and $r \sqsubseteq r'$ do
12:         if $t''(w) \preceq t''(w')$ then
13:           nextwhile;
14:         end if
15:       end for
16:       $\mathcal{R}' := \mathcal{R}' \cup (sh_\downarrow(r) \cap \mathcal{R})$; $\mathcal{R}'' := \mathcal{R}'' \cup \{r\}$;
17:     end if
18:   end while
19:   $\mathcal{M} := \mathcal{M} \cup \{(t^*, \mathcal{R}'') \mid t^* \in \rho(t'), \mathcal{R}'' \neq \emptyset\}$;
20: end while
21: return $\mathcal{T}$;
```

with t'. Finally, once $\mathcal{R}'$ is empty, in Line 19 we add all tupels $(t^*, \mathcal{R}'')$ to $\mathcal{M}$, where t^* is obtained by recursively calling ρ on t'.

Example 2. ***[ToDo: altes Beispiel:]*** Let $\mathcal{K}$ be the knowledge base from Example 1 and let $\mathcal{K}' = \mathcal{K} \setminus \{domain(\texttt{has\_pet}) = \texttt{Animal}\}$. Figure 3.1 depicts the set of all trees in $\rho(\texttt{Human} \sqcap \exists\texttt{has.Animal})$ w.r.t. $\mathcal{K}'$.

Example 3. ***[ToDo: Beispiel aus Diss:]*** Let $\mathcal{K}$ be the following knowledge base:

$$N_C = \{\texttt{Human}, \texttt{Animal}, \texttt{Bird}, \texttt{Cat}\}$$
$$N_R = \{\texttt{has}, \texttt{has\_child}, \texttt{has\_pet}\}$$
$$\mathcal{K} = \{\texttt{has\_pet} \sqsubseteq \texttt{has}, \texttt{has\_child} \sqsubseteq \texttt{has}, \texttt{Bird} \sqsubseteq \texttt{Animal}, \texttt{Cat} \sqsubseteq \texttt{Animal}\}$$

Proposition 1. *ρ is a is a finite, proper and weakly complete downward refinement operator on $(T_{min}, \preceq)$.*

Proof. In the following, let $t \in T_{min}$ and $v \in V(t)$.

First, it is easily seen that ρ is a downward refinement operator. Every operation of ρ adds a label or a subtree to a node v, or replaces a label or edge-label by a refined label or edge respectively. Hence, $t' \preceq t$ for all $t' \in \rho(t)$.

Regarding finiteness of ρ, the first part of Lemma 2 guarantees that there is only a finite number of minimal $\mathcal{EL}$ trees up to a fixed depth. It then follows

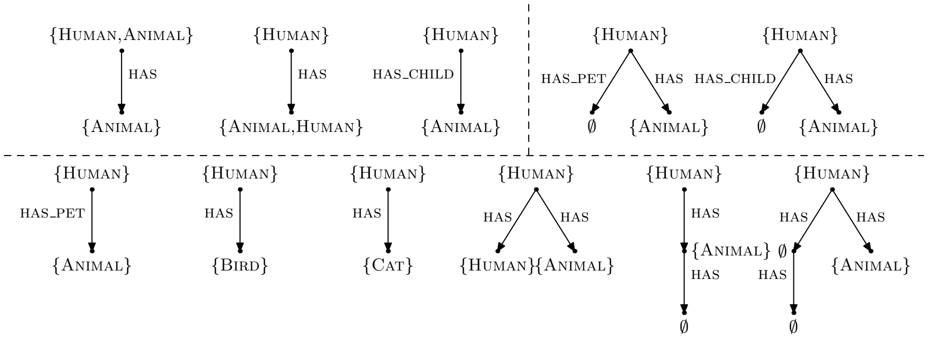

Fig. 3. The set $\rho(\texttt{Human} \sqcap \exists \texttt{has.Animal})$ of minimal trees w.r.t. the knowledge base $\mathcal{K}'$ from Example 3.

from the second part of Lemma 2 that for a given tree t, $\rho(t)$ only consists of trees of depth at most $depth(t) + 1$. Hence, $\rho(t)$ is finite.

In order to prove properness of ρ, it is sufficient to show $t \not\preceq t'$ for $t' \in \rho(t)$. To the contrary, assume $t \preceq t'$ and that t has been refined at v. Let $\mathcal{S}$ be a simulation from t' to t. Since v has been refined, it follows that $(v, v) \notin \mathcal{S}$. We have that $\mathcal{S}$ is a simulation, so there must be some $v' \in V(t)$ with $level(v') = level(v)$ such that $(v', v) \in \mathcal{S}$. This implies that there is a simulation $\mathcal{S}'$ on t' with $\{(v', v), (v, v)\} \subseteq \mathcal{S}'$. It follows that t' contains a redundant subtree at the predecessor of v, contradicting to the minimality of t'.

Regarding weakly completeness, let $depth(t) \leq n$. We show that t is reachable from $t_\top$ by nested induction on n and $m := |\{(root(t), r, w) \in E(t)\}|$. For the induction base case $n = 0, m = 0$, t is just a single node labeled with some concept names. It is easily seen that by repeatedly applying $e\ell(t, v)$ and $r\ell(t, v)$ to this node we eventually reach t. For the induction step, let $n > 0, m > 0$. Hence, the root of t is has m successor nodes $w_1, \ldots, w_m$ attached along edges $r_1, \ldots, r_m$ to t. By the induction hypothesis, the tree t_{m-1}, which is obtained from t by removing the subtree $t(w_1)$ from t, is reachable from $t_\top$. Also, there is a refinement chain θ from $t_\top$ to $t(w_1)$ such that an intermediate tree t'_{w_1} occurs in θ and $t' = t_{m-1}[root(t) \leftarrow (r'_1, t'_{w_1})] \in as(t_{m-1}, root(t))$ for some r'_1 with $r_1 \sqsubseteq r'_1$. Hence, we can first reach t' from $t_\top$ and then, by applying the remaining refinement steps from θ to t' and refining r'_1 to r_1, eventually reach t.

Still, ρ is not ideal, since it is not complete. It is however easy to derive a complete operator ρ^* from ρ:

$$\begin{aligned}\rho^*(t) := max\{t' \mid t_\top \rightsquigarrow^*_\rho t', t' \prec t \text{ and}\\ depth(t') \leq depth(t) + 1\}.\end{aligned}$$

This construction is needed, because we would for example not be able to reach $\exists r.(A_1 \sqcap A_2)$ starting from $\exists r.A_1 \sqcap \exists r.A_2$ with ρ.

Theorem 1. *The $\mathcal{EL}$ downward refinement operator ρ^* is ideal.*

Remark 1. In [12] it has been shown that for languages other than $\mathcal{EL}$ complete and non-redundant refinement operators do not exist (under a mild assumption). The same result carries over to our setting:

Proposition 2. *Let $\psi : T_{min} \rightarrow \mathcal{P}(T_{min})$ be a complete refinement operator. Then ψ is redundant.*

Proof. We assume $\mathcal{K} = \emptyset$ and N_C contains A_1 and A_2. Since ψ is complete and its refinements are minimal, we have $\top \rightsquigarrow^* A_1$. Similarly, $\top \rightsquigarrow^* A_1$, $A_1 \rightsquigarrow^* A_1 \sqcap A_2$, and $A_2 \rightsquigarrow^* A_1 \sqcap A_2$. We have $A_1 \not\sqsubseteq A_2$ and $A_2 \not\sqsubseteq A_1$, which means that $A_1 \not\rightsquigarrow^* A_2$ and $A_2 \not\rightsquigarrow^* A_1$. Hence, $A_1 \sqcap A_2$ can be reached from $\top$ via a refinement chain going through A_1 and a different refinement chain not going through A_1, i.e. ψ is redundant.

3.2 Optimisations

We used two different kinds of optimisations: The first is concerned with the performance of minimality tests and the second reduces the number of trees returned by ρ by incorporating more background knowledge.

Recall from Section 2.3 that checking for minimality of a tree t involves computing a maximal simulation $\mathcal{S}$ on $V(t)$ and is in $\mathcal{O}(|t|^2)$. In order to avoid expensive re-computations of $\mathcal{S}$ after each refinement step, the data-structure of t is extended such that sets $\mathcal{C}_1^{\leftarrow}(v)$, $\mathcal{C}_1^{\rightarrow}(v)$, $\mathcal{C}_2^{\leftarrow}(v)$ and $\mathcal{C}_2^{\rightarrow}(v)$ are attached to every node $v \in V(t)$. Here, the set $\mathcal{C}_1^{\leftarrow}(v)$ contains those nodes w such that (SC1) holds for (v, w) according to Definition 1. Likewise, $\mathcal{C}_2^{\rightarrow}(v)$ is the set of those nodes w such that (SC2) holds for (w, v), and $\mathcal{C}_1^{\leftarrow}(v)$ and $\mathcal{C}_2^{\rightarrow}(w)$ are defined accordingly. When checking for minimality, it is moreover sufficient that each such set is restricted to only consist of nodes from *onlevel*(v) excluding v itself. This fragmentation of $\mathcal{S}$ allows us to perform local updates instead of re-computation of $\mathcal{S}$ after an operation is performed on v. For example, when the label of v is extended, we only need to recompute $\mathcal{C}_1^{\leftarrow}(v)$, update $\mathcal{C}_1^{\rightarrow}(w)$ for every $w \in \mathcal{C}_1^{\leftarrow}(v)$, and then repeatedly update $\mathcal{C}_2^{\rightarrow}(v')$ and $\mathcal{C}_2^{\leftarrow}(v')$ for every predecessor node v' of an updated node until we reach the root of t. This method saves a considerable amount of computation, since the number of nodes affected by an operation is empirically relatively small.

In order to keep $|\rho(t)|$ small, we use role domains and ranges as well as disjoint concepts inferred from $\mathcal{K}$. The domain restriction axioms can be used to reduce the set of role names considered when adding a subtree or refining an edge: For instance, let w be a node, (v, r, w) the edge pointing to w, and *range*$(r) = A$. When adding an edge (w, s, u), we ensure that *range*$(r) \sqcap$ *domain*(s) is satisfiable. This ensures that only compatible roles are combined. Similar effects are achieved by mode declarations in ILP tools. However, in OWL ontologies role domains and ranges are usually already present and do not need to be added manually. Similar optimisations can be applied to edge refinement. In *as*(t, v), we furthermore use range restrictions to automatically label a new node with the corresponding role range. For example, if the edge has label r and *range*$(r) = A$, then the new node w is assigned label $\ell(w) = \{A\}$ (instead of $\ell(w) = \emptyset$).

We now address the optimisation of extending node labels in the implementation of the function $e\ell$. Let A be a concept name for which we want to know whether or not we can add it to $\ell(v)$. We first check $A \sqsubseteq \ell(v)$. If yes, we discard A since we could reach an equivalent concept by refining a concept in $\ell(v)$, i.e. we perform redundancy reduction. Let (u, r, v) be the edge pointing to v and $range(r) = B$. We verify that $A \sqcap B$ is satisfiable and discard A otherwise. Additionally as before, we test whether $\ell(v) \sqsubseteq A$. If yes, then A is also discarded, because adding it would not result in a proper refinement. Performing the last step in a top down manner, i.e. start with the most general concepts A in the class hierarchy, ensures that we compute the maximum of eligible concepts, which can be added to $\ell(v)$. In summary, we make sure that the tree we obtain is label reduced, and perform an on-the-fly test for the satisfiability of its corresponding concept. Applying similar ideas to the case of label refinement is straight forward.

In practice, the techniques briefly described in this section narrow the set of trees returned in a refinement step significantly by ruling out concepts, which are unsatisfiable w.r.t. $\mathcal{K}$ or which can also be reached via other refinement chains. This is is illustrated by the following example.

Example 4. Let $\mathcal{K}$ be as in Example 1 and define $\mathcal{K}' := \mathcal{K} \cup \{domain(\texttt{has\_child}) = \texttt{Human}, range(\texttt{has\_child}) = \texttt{Human}, \texttt{Human} \sqcap \texttt{Animal} \equiv \bot\}$. By incorporating the additional axioms, $\rho(\texttt{Human} \sqcap \exists\texttt{has.Animal})$ only contains the trees on the right-hand side of the dashed line in Figure 3, except for $\texttt{Human} \sqcap \exists\texttt{has\_child}.\top \sqcap \exists\texttt{has.Animal}$, which becomes $\texttt{Human} \sqcap \exists\texttt{has\_child.Human} \sqcap \exists\texttt{has.Animal}$ due to the range of $\texttt{has\_child}$.

4 Evaluation of the Operator

In order to evaluate the operator, we computed *random refinement chains* of ρ. A random refinement chain is obtained by applying ρ to $\top$, choosing one of the refinements uniformly at random, then applying ρ to this refinement, etc.

In order to asses the performance of the operator, we tested it on real ontologies chosen from the TONES repository[5], including some of the most complex OWL ontologies. We generated 100 random refinement chains of length 8 and measured the results. We found experimentally that this allows us to evaluate the refinement operator on a diverse set of concept trees. The tests were run on an Athlon XP 4200+ (dual core 2.2 GHz) with 4 GB RAM. As a reasoner we used Pellet 1.5. The benchmarks do not include the time to load the ontology into the reasoner and classify it.

The results are shown in Table 3. The first four columns contain the name and relevant statistics of the ontology considered. The next column shows the average time the operator needed on each input concept. In the following column this value is divided by the number of refinements of the input concept. The subsequent column shows how much time is spend on reasoning during the

[5] http://owl.cs.manchester.ac.uk/repository/

Name	Logical axioms	Classes	Roles	ρ av. time (in ms)	ρ per ref. (in ms)	Reasoning time (%)	Refinements (av. and max.)		Ref. size (av. and max.)	
Genes	42656	26225	4	167.2	0.14	68.4	1161.5	2317	5.0	8
CTON	33203	17033	43	76.2	0.08	5.1	220.2	28761	5.8	24
Galen	4940	2748	413	3.5	0.21	37.1	17.0	346	4.9	16
Process	2578	1537	102	193.6	0.16	27.2	986.5	23012	5.7	22
Transport	1157	445	89	164.4	0.09	5.9	985.2	22651	5.7	24
Earthrealm	931	559	81	407.4	0.17	23.2	1710.3	27163	5.7	19
Tambis	595	395	100	141.6	0.09	1.5	642.4	26685	5.8	23

Table 3. Benchmark results on ontologies from the TONES repository. The results show that ρ works well even on large knowledge bases. The time needed to compute a refinement is below one millisecond and does not show large variations.

computation of refinements. The two last columns contain the number of refinements obtained and their size. Here, we measure size as the number of nodes in a concept tree plus the sum of the cardinality of all node labels.

The most interesting insight from Table 3 is that despite the different size and complexity of the ontologies, the time needed to compute a refinement is low and does not show large variations (between 0.09 and 0.21 ms). This indicates that the operator scales well to large knowledge bases. It can also be observed that the number of refinements can be very high in certain cases, which is due to the large number of classes and properties in many ontologies and the absence of explicit or implicit disjointness between classes. We want to note that when the operator is used to learn concepts from instances (standard learning task), one can use the optimisations in Section 3.2 and consider classes without common instances instead of class disjointness. In this case, the number of refinements of a given concept will usually be much lower, since no explicit disjointness axioms are required. In all experiments we also note that the time the reasoner requires differs a lot (from 1.5% to 68.4%). However, since the number of reasoner requests is finite and the results are cached, this ratio will decrease with more calls to the refinement operator. Summing up, the results show that efficient ideal refinement on large ontologies can be achieved in $\mathcal{EL}$, which in turn is promising for $\mathcal{EL}$ concept learning algorithms.

5 Related Work

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. [15]). The investigated operators are usually based on horn clauses. In general, applying such operators to DL problems is considered not be a good choice [3]. However, some of 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.

In Shapiro's Model Inference System [17], he describes how refinement operators can be used to adapt a hypothesis to a sequence of examples. In the following years, refinement operators became widely used. [19] 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. The results have been used to show the non-existence of ideal refinement operators for clauses ordered by θ-subsumption. Later, refinement operators have been extended to theories (clause sets) [8].

Within the last decade, several refinement operators for DLs have been investigated. The most fundamental work is [12], which shows for many description languages the maximal sets of properties which can be combined. Among other things, a non-ideality result for the languages $\mathcal{ALC}$, $\mathcal{SHOIN}$, and $\mathcal{SROIQ}$ is shown. We extend this work by providing an ideality result for $\mathcal{EL}$. Refinement operators for $\mathcal{ALER}$ [3], $\mathcal{ALN}$ [7], $\mathcal{ALC}$ [13,9] have been created and used in learning algorithms. It has been stated in [6] and [7] that further research into refinement operator properties is required for building the theoretical foundations of learning in DLs. Finally, [14] provides ideal refinement in $\mathcal{AL}$-log, a hybrid language merging Datalog and $\mathcal{ALC}$, but naturally a different order than DL subsumption was used.

6 Conclusions and Future Work

In summary, we have provided an efficient ideal $\mathcal{EL}$ refinement operator, thereby closing a gap in refinement operator research. We have shown that the operator can be applied to very large ontologies and makes profound use of background knowledge. In future work, we want to incorporate the refinement operator in learning algorithms, and investigate whether certain extensions of $\mathcal{EL}$ may be supported by the operator without losing ideality.

References

1. F. Baader, R. Molitor, and S. Tobies. Tractable and decidable fragments of conceptual graphs. In *Seventh International Conference on Conceptual Structures (ICCS'99)*, number 1640 in LNCS, pages 480–493. Springer Verlag, 1999.
2. F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. Applied Logic*, 5(3):392–420, 2007.
3. 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.

4. Olivier Bodenreider, Barry Smith, Anand Kumar, and Anita Burgun. Investigating subsumption in SNOMED CT: An exploration into large description logic-based biomedical terminologies. *Artificial Intelligence in Medicine*, 39(3):183–195, 2007.
5. Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini, editors. *Ontology Learning from Text: Methods, Evaluation and Applications*, volume 123 of *Frontiers in Artificial Intelligence*. IOS Press, JUL 2007.
6. Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-intensive induction of terminologies from metadata. In *Third International Semantic Web Conference*, pages 441–455. Springer, 2004.
7. Nicola Fanizzi, Stefano Ferilli, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Downward refinement in the ALN description logic. In *HIS*, pages 68–73. IEEE Computer Society, 2004.
8. Nicola Fanizzi, Stefano Ferilli, Nicola Di Mauro, and Teresa Maria Altomare Basile. Spaces of theories with ideal refinement operators. In Georg Gottlob and Toby Walsh, editors, *Proc. of 18th Int. Joint Conf. on Artificial Intelligence*, pages 527–532. Morgan Kaufmann, 2003.
9. Luigi Iannone, Ignazio Palmisano, and Nicola Fanizzi. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159, 2007.
10. Ralf Küsters. *Non-standard inferences in description logics.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.
11. Jens Lehmann. Hybrid learning of ontology classes. In *Machine Learning and Data Mining in Pattern Recognition, 5th International Conference*, volume 4571 of *Lecture Notes in Computer Science*, pages 883–898. Springer, 2007.
12. Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *Proc. of 17th Int. Conf. on Inductive Logic Programming (ILP 2007)*, volume 4894 of *LNCS*, pages 161–174. Springer, 2008. Best Student Paper.
13. Jens Lehmann and Pascal Hitzler. A refinement operator based learning algorithm for the ALC description logic. In *Proc. of 17th Int. Conf. on Inductive Logic Programming (ILP 2007)*, volume 4894 of *LNCS*, pages 147–160. Springer, 2008. Best Student Paper.
14. Francesca A. Lisi and Donato Malerba. Ideal refinement of descriptions in AL-log. In Tamás Horváth, editor, *Proc. of 13th Int. Conf. on Inductive Logic Programming (ILP)*, volume 2835 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2003.
15. Shan-Hwei Nienhuys-Cheng and Ronald de Wolf, editors. *Foundations of Inductive Logic Programming.* Lecture Notes in Computer Science. Springer, 1997.
16. Prakash Ramanan. Efficient algorithms for minimizing tree pattern queries. In *SIGMOD '02: Proc. of the 2002 ACM SIGMOD Int. Conf. on Management of data*, pages 299–309. ACM, 2002.
17. E. Y. Shapiro. Inductive inference of theories from facts. In J. L. Lassez and G. D. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 199–255. The MIT Press, 1991.
18. The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000.
19. P. R. J. van der Laag and S-H. Nienhuys-Cheng. Existence and nonexistence of complete refinement operators. In *Proc. of 7th Europ. Conf. on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag, 1994.