From "Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference"

Learning the CLASSIC Description Logic: Theoretical and Experimental Results

William W. Cohen
AI Principles Research Department
AT&T Bell Laboratories
Murray Hill, NJ 07974
wcohen@research.att.com

Haym Hirsh*
AI Principles Research Department
AT&T Bell Laboratories
Murray Hill, NJ 07974
hirsh@research.att.com

Abstract

We present a series of theoretical and experimental results on the learnability of description logics. We first extend previous formal learnability results on simple description logics to C-CLASSIC, a description logic expressive enough to be practically useful. We then experimentally evaluate two extensions of a learning algorithm suggested by the formal analysis. The first extension learns C-CLASSIC descriptions from individuals. (The formal results assume that examples are themselves descriptions.) The second extension learns disjunctions of C-CLASSIC descriptions from individuals. The experiments, which were conducted using several hundred target concepts from a number of domains, indicate that both extensions reliably learn complex natural concepts.

1 INTRODUCTION

One well-known family of formalisms for representing knowledge are *description logics*, sometimes also called *terminological logics* or *KL-ONE-type languages*. Description logics have been applied in a number of contexts [Beck *et al.*, 1989; Devanbu *et al.*, 1991; Mays *et al.*, 1987; Wright *et al.*, 1993]; additionally, the complexity of deductive reasoning using description logics is fairly well understood.

Recently we have begun to analyze the complexity of using description logics to support *inductive* reasoning—*i.e.*, learning. Our analysis has focused on determining which description logics are learnable in Valiant's [1984] model of *pac-learnability* [Cohen and Hirsh, 1992b], and with understanding the complexity of the operations necessary to support learning [Cohen *et al.*, 1992].

*Also at Computer Science Department, Rutgers University, New Brunswick, NJ 08903.

In this paper, we build on these formal results in several ways. We extend the previous formal results to the description logic C-CLASSIC, which is expressive enough to be practically useful. We also present two extensions of an algorithm suggested by the formal results: the first extension learns descriptions from individuals, and the second learns disjunctions of descriptions from individuals. Finally, we experimentally evaluate these two extensions. Experiments conducted using several hundred naturally occurring concepts from a number of domains support the claim that both extensions can reliably learn complex, naturally-occurring concepts.

2 BACKGROUND

2.1 DESCRIPTION LOGICS

CLASSIC is a knowledge representation system based on a *description logic* (henceforth DL). Some recent surveys of work in description logics can be found in [MacGregor, 1991; Woods and Schmolze, 1992]; however to keep this paper self-contained we will give a brief review below.

Description logics are a family of formalisms for representing knowledge. DLs trace their ancestry back to semantic nets and frame-based languages, but place a stronger emphasis on clear formal semantics and provably tractable inference.

DLs are used to reason about sets of atomic elements called *individuals*; in particular, DLs are used to construct *descriptions* of sets of individuals and then to reason about these descriptions. Descriptions are typically defined compositionally using description *constructors* and building blocks known as *primitives* and *roles*. A *primitive* denotes a specific set of individuals. A *role* denotes a specific binary relation between individuals. Constructors are typically operators like `AND` or `SOME`, which we will write in a prefix notation.

Descriptions are built up by specifying constraints on properties an individual must have. As an example, in

Table 1: Description Logic Constructors

Constructor	Semantics
AND	$\mathcal{I}(\texttt{(AND D}_1 \ \ldots \ \texttt{D}_n\texttt{)}) = \cap_{i=1}^{n} \mathcal{I}(\texttt{D}_i)$
ALL	$\mathcal{I}(\texttt{(ALL r D)}) = \{x \in \Delta : \forall y \ \langle x, y\rangle \in \mathcal{I}(\texttt{r}) \Rightarrow y \in \mathcal{I}(\texttt{D})\}$
SOME	$\mathcal{I}(\texttt{(SOME r)}) = \{x \in \Delta : \exists y \ \langle x, y\rangle \in \mathcal{I}(\texttt{r})\}$
SOMEC	$\mathcal{I}(\texttt{(SOMEC r D)}) = \{x \in \Delta : \exists y \ \langle x, y\rangle \in \mathcal{I}(\texttt{r}) \wedge y \in \mathcal{I}(\texttt{D})\}$
AT-LEAST	$\mathcal{I}(\texttt{(AT-LEAST n r)}) = \{x \in \Delta : \lvert\{y : \langle x, y\rangle \in \mathcal{I}(\texttt{r})\}\rvert \geq n\}$
AT-MOST	$\mathcal{I}(\texttt{(AT-MOST n r)}) = \{x \in \Delta : \lvert\{y : \langle x, y\rangle \in \mathcal{I}(\texttt{r})\}\rvert \leq n\}$
MIN	$\mathcal{I}(\texttt{(MIN u)}) = \{x \in \Delta : x \text{ is a real number and } x \geq u\}$
MAX	$\mathcal{I}(\texttt{(MAX u)}) = \{x \in \Delta : x \text{ is a real number and } x \leq u\}$
ONE-OF	$\mathcal{I}(\texttt{(ONE-OF I}_1 \ldots \texttt{I}_n\texttt{)}) = \{x \mid x \in \mathcal{I}(\texttt{I}_1) \vee \ldots \vee x \in \mathcal{I}(\texttt{I}_n)\}$
FILLS	$\mathcal{I}(\texttt{(FILLS r I}_1 \ldots \texttt{I}_n\texttt{)}) = \{x \mid \forall j : 1 \leq j \leq n, \ \exists z \ [(x, z) \in \mathcal{I}(\texttt{r}) \wedge z \in \texttt{I}_j]\}$
SAME-AS	$\mathcal{I}(\texttt{(SAME-AS (a}_1 \ \ldots \ \texttt{a}_k\texttt{) (b}_1 \ \ldots \ \texttt{b}_l\texttt{))}) = \{x \in \Delta : \mathcal{I}(\texttt{a}_k) \circ \ldots \circ \mathcal{I}(\texttt{a}_1)(x) = \mathcal{I}(\texttt{b}_l) \circ \ldots \circ \mathcal{I}(\texttt{b}_1)(x)\}$
THING	$\mathcal{I}(\texttt{THING}) = \Delta$
NOTHING	$\mathcal{I}(\texttt{NOTHING}) = \emptyset$

a description logic with the constructors **AND**, **ALL** and **SOME**, one might use the description

```
(AND family
   (ALL husband (AND retired (ALL age over-65)))
   (ALL wife employed)
   (SOME child (AND student (ALL school graduate))))
```

to denote the set of families where the husband is retired, the wife is employed, and some child is attending graduate school. In the example, `family`, `retired`, `over-65`, `student` and `graduate` are primitives, and `husband`, `wife`, `child` and `school` are roles.

More formally, a *description* is a representation of a subset of some domain Δ of atomic individuals. A primitive symbol $\texttt{p}_i$ is a description denoting a subset of Δ; we will write this subset as $\mathcal{I}(\texttt{p}_i)$. If $\texttt{D}_1 \ \ldots \ \texttt{D}_n$ are descriptions, then $\texttt{(AND D}_1 \ \ldots \ \texttt{D}_n\texttt{)}$ is a description representing the set

$$\mathcal{I}(\texttt{(AND D}_1 \ \ldots \ \texttt{D}_n\texttt{)}) = \cap_{i=1}^{n} \mathcal{I}(\texttt{D}_i)$$

Using the same sort of recursive definition, one can define other constructors easily. Table 1 presents some common constructors, together with their semantics. In the table, `r` is always a *role*; a role `r` denotes a subset of $\Delta \times \Delta$, which is written $\mathcal{I}(\texttt{r})$. $\texttt{I}_j$ is always an *individual*. Finally, `n` is always an integer and `u` is always a real number. We assume that Δ contains the real numbers. Note that the semantics of **FILLS** and **ONE-OF**, as given in the table, are somewhat nonstandard. For somewhat technical reasons the individuals used as arguments to the **FILLS** and **ONE-OF** constructors are defined to be disjoint subsets of the domain, rather than domain elements, as is more usually the case.[1]

[1] In a DL with individuals that are domain elements, a description like `(AND (ALL Car (ONE-OF Saab Volvo)) (ALL Car Yuppiemobile))` would imply the disjunctive fact that either `Saabs` or `Volvos` are `Yuppiemobiles`. Reasoning with such disjunctive information is intractable. Using the modified semantics `Saab` and `Volvo` would stand for two disjoint sets of objects, rather than two distinct objects; tractable and complete inference procedures exist for this modified semantics [Borgida and Patel-Schneider, 1992].

In this paper we focus on a particular description logic called CLASSIC2. CLASSIC2 is a reimplementation and slight extension of CLASSIC1 [Borgida *et al.*, 1989; Brachman, 1990] that contains all of the constructors summarized in Table 1. The main extensions to the logic relative to CLASSIC1 are the **MIN** and **MAX** constructors, and the addition of role hierarchies and role inverses; the other constructors are inherited from CLASSIC1.

Most of our results actually concern the DL with the constructors **AND**, **ALL**, **AT-LEAST**, **AT-MOST**, **FILLS**, **ONE-OF**, **MIN**, and **MAX**—*i.e.*, CLASSIC2 without the **SAME-AS** constructor or role hierarchies. In the remainder of this paper we will call this DL C-CLASSIC. A knowledge-based management based on C-CLASSIC has been used for a number of real-world applications (*e.g.*, [Wright *et al.*, 1993]).

2.2 REASONING IN CLASSIC

DLs are primarily used for taxonomic reasoning, and hence an important operation is determining if one description is more general than another. The generality relationship used for descriptions is called *subsumption*: description $\texttt{D}_1$ is said to *subsume* $\texttt{D}_2$ if $\mathcal{I}(\texttt{D}_1) \supseteq \mathcal{I}(\texttt{D}_2)$ for every possible definition of the primitives and roles appearing in $\texttt{D}_1$ and $\texttt{D}_2$. Subsumption is thus closely related to the familiar notion of set inclusion.

Subsumption in CLASSIC is fairly well understood. The subsumption algorithms for CLASSIC2 are similar to those for CLASSIC1—descriptions are first *normalized* by converting them to a labeled graph structure called a *description graph*, and then subsumption can be efficiently tested by graph-matching operations. Below we will briefly review the special case of description graphs that occur when the standard CLASSIC2

```
(AND
 Family
 (ALL Husband
      (AND Student
           (ALL Age (ONE-OF 31 32 33))))
 (ALL Wife
      (AND Employed
           (FILLS employer IBM)))
 (AT-MOST 2 Child)
 (AT-LEAST 2 Car)
 (ALL Car
      (ALL Maker (ONE-OF Volvo Saab))))
```

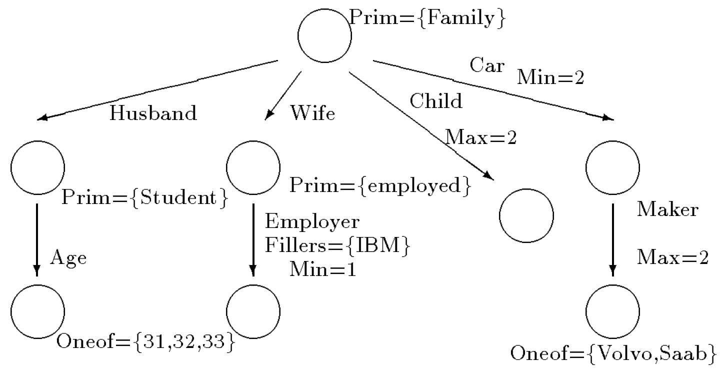


Figure 1: A C-Classic Description and Description Graph

algorithms are applied to C-Classic.

For C-Classic, description graphs are always trees. The nodes of *description trees* are labeled with tuples (*dom*, *prim*, *mn*, *mx*). The *dom* component is either a set of individuals $\{\mathtt{I}_1, \ldots, \mathtt{I}_n\}$, which intuitively represents the constraint that the condition (ONE-OF $\mathtt{I}_1$ $\ldots \mathtt{I}_n$) must hold at this vertex, or the special symbol UNIV, which indicates that no ONE-OF restriction is in force. The *mn* and *mx* labels are either real numbers, representing MIN and MAX restrictions, or the symbol NOLIMIT, again representing the absence of any restriction. The *prim* label is a set of primitive concepts.

Each edge in a description graphs is labeled with a tuple ($\mathtt{r}$, *least*, *most*, *fillers*). Intuitively *least* is an integer representing an AT-LEAST restriction on the role $\mathtt{r}$, *most* is an integer representing an AT-MOST restriction on $\mathtt{r}$, and *fillers* is a set of individuals representing a FILLS restriction on $\mathtt{r}$. To allow for an absent AT-MOST restriction, *most* can also be the special symbol NOLIMIT.

More formally, given a vertex v from a description tree T, with label (*dom*, *prim*, *mn*, *mx*), a domain element x is defined to be in the *extension of* v iff the following all hold.

- If $dom = \{\mathtt{I}_1, \ldots, \mathtt{I}_k\}$ (*i.e.*, if $dom \neq$ UNIV) then $x \in \mathcal{I}((\mathtt{ONE\text{-}OF}\ \mathtt{I}_1 \ldots \mathtt{I}_k))$.
- For each $\mathtt{p}_i \in prim$, $x \in \mathcal{I}(\mathtt{p}_i)$.
- If $mn \neq$ NOLIMIT, $x \in \mathcal{I}((\mathtt{MIN}\ \ mn))$.
- If $mx \neq$ NOLIMIT, $x \in \mathcal{I}((\mathtt{MAX}\ \ mx))$.
- For each edge from v to w with label ($\mathtt{r}$, *least*, *most*, *fillers*) the following all hold:
  - if y is any domain element such that $(x, y) \in \mathcal{I}(\mathtt{r})$, then y is in the extension of w;
  - $least \leq |\{y|(x,y) \in \mathcal{I}(\mathtt{r})\}|$,
  - if $most \neq$ NOLIMIT then $|\{y|(x,y) \in \mathcal{I}(\mathtt{r})\}| \leq most$,
  - if $fillers = \{\mathtt{I}_1, \ldots, \mathtt{I}_k\}$, then $x \in \mathcal{I}((\mathtt{FILLS}\ \mathtt{I}_1 \ldots \mathtt{I}_k))$

Finally, an individual x is in the extension of the description tree T iff it is in the extension of the root vertex of T.

To normalize a C-Classic description, the description is first converted to a description tree using a simple recursive algorithm. The description tree is then converted into a *canonical form* by further normalization operators: for example, one operator looks for edges labeled ($\mathtt{r}$, *least*, *most*, *fillers*) where $|fillers| > least$, and then replaces each such *least* label with $|fillers|$. Figure 1 contains an example of a C-Classic description, and the equivalent canonical description tree. (To simplify the diagram, vacuous labels like *dom*=UNIV and *least*=0 are not shown.) For a more complete discussion of Classic description graphs, and the semantics for Classic, consult Borgida and Patel-Schneider [1992].

2.3 PAC-LEARNABILITY

The problem of inductive learning is to extrapolate a general description of a concept c from a set of *training examples*—things that have been labeled by an oracle as positive if they are elements of c and negative otherwise. To formalize this, let X refer to a *domain*—a set of things that might serve as positive or negative examples. A *concept* c is a subset of X. A *concept class* is a set of concepts; this will designate a constrained set of "target" concepts that could be labeling the training data. Associated with each concept class is a language $\mathcal{L}$ for writing down concepts in that class. In this paper the representation in $\mathcal{L}$ for the concept c will also be denoted c (as it will be clear from context whether we refer to the concept or its representation). We will also assume the existence of some measure for the size of a representation of a concept. Typically this measure will be polynomially related to the number of bits needed to write down a concept.

In learning, the goal is to find the unknown target concept $c \in \mathcal{L}$ (or some reasonable approximation thereof) from a set of labeled examples. Usually examples are elements of the domain X, with $x \in X$ labeled as positive if $x \in c$ and negative otherwise. We will depart from this model here, and instead assume that examples are *concepts* selected from $\mathcal{L}$, and that an example $x \in \mathcal{L}$ will be labeled as positive if c subsumes x, and labeled as negative otherwise. (Thus in learning DLs both examples and target concepts will be descriptions.) This *single-representation trick* [Dietterich *et al.*, 1982] has been used in comparable situations in the computational learning theory literature (*e.g.*, [Haussler, 1989]). In analyzing first-order languages it is particularly useful because there is often no standard representation for instances.

Our model of "efficient learnability" is based on Valiant's model of pac-learnability [Valiant, 1984]. We assume a static probability distribution by which examples are drawn, *i.e.*, some distribution P over the language $\mathcal{L}$. The probability distribution P gives us a natural way of measuring the quality of a hypothesis h; one can simply measure the probability that h's label will disagree with the label of the target concept c on an example chosen according to P. The goal of pac-learning is to produce a hypothesis that will with high probability score well according to this measure—that is, a hypothesis that will be "probably approximately correct"—regardless of the probability distribution P and the target concept $c \in \mathcal{L}$.

More formally, let a *sample of* c be a pair of multisets S^+ and S^- drawn from $\mathcal{L}$ according to P, with S^+ containing the positive examples of c and S^- containing the negative examples. Let $error_{P,c}(h)$ be the probability that h and c disagree on an example x drawn randomly according to P (*i.e.*, the probability that h subsumes x and c does not subsume x, or that c subsumes x and h does not subsume x). Let $\mathcal{L}_n$ denote the set of concepts $c \in \mathcal{L}$ of size no greater than n.

A language $\mathcal{L}$ is said to be *pac-learnable* iff there is an algorithm Learn and a polynomial function $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_t)$ so that for every $n_t > 0$, every $c \in \mathcal{L}_{n_t}$, every $\epsilon : 0 < \epsilon < 1$, every $\delta : 0 < \delta < 1$, and every probability distribution P over $\mathcal{L}_{n_e}$, when Learn is run on a sample of size $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_t)$ or larger it takes time polynomial in the sample size and outputs a concept $h \in \mathcal{L}$ for which $Prob(error_{P,c}(h) > \epsilon) < \delta$.

In other words, even given adversarial choices of n_t, n_e, ϵ, δ, P, and $c \in \mathcal{L}_{n_t}$, Learn will with high confidence return a hypothesis h that is approximately correct (with respect to the correct hypothesis c and the distribution of examples P), using only polynomial time and a polynomial number of examples. The polynomial bound $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_t)$ on the number of examples is called the *sample complexity* of Learn.

As noted above, this formalization is conventional, except for the assumption that examples are descriptions that are marked positive when subsumed by the target concept. In the discussions below, the *standard pac-learning model* refers to the variant of this model resulting when examples are domain elements.

2.4 RELEVANT PREVIOUS RESULTS

Only a few previous papers have directly addressed the the pac-learnability of description logics. However, a connection can be drawn between pac-learnability and certain previous formal results on the complexity of reasoning in description logics.

For instance, it is known that if $\mathcal{L}$ is pac-learnable in the standard model, then $\mathcal{L} \in P/Poly$ [Schapire, 1990], where $P/Poly$ is the set of languages accepted by a (possibly nonuniform) family of polynomial-size deterministic circuits. This result can be used to obtain a number negative results in our model, such as the following:

Theorem 1 *In the model defined in Section 2.3, if concepts in a language $\mathcal{L}$ can be represented as strings over $\{0,1\}$ with only a polynomial increase in size, and if subsumption for $\mathcal{L}$ is either NP-hard or coNP-hard, then $\mathcal{L}$ is not pac-learnable unless $NP \subseteq P/Poly$.*

Proof: For any language $\mathcal{L}$ and a concept $c \in \mathcal{L}$, let $\hat{c}$ be a concept that has the same representation as c, but which denotes the set

$$\{d \in \mathcal{L} : d \text{ is subsumed by } c\}$$

Also define $\hat{\mathcal{L}} \equiv \{\hat{c} : c \in \mathcal{L}\}$. It is immediate that $\mathcal{L}$ is pac-learnable in the model of Section 2.3 iff $\hat{\mathcal{L}}$ is pac-learnable in the standard model, and that testing membership for a concept $\hat{c} \in \hat{\mathcal{L}}$ is as hard as testing subsumption for the concept $c \in \mathcal{L}$. By Theorem 7 of Schapire [1990], if $\hat{\mathcal{L}}$ is pac-learnable then $\hat{\mathcal{L}} \in P/Poly$; thus if $\hat{\mathcal{L}}$ is NP-hard ($coNP$-hard) it follows that $NP \subseteq P/Poly$ ($coNP \subseteq P/Poly$). Finally since $P/Poly$ is closed under complementation, $coNP \subseteq P/Poly$ implies that $NP \subseteq P/Poly$. ■

This theorem immediately establishes the non-learnability of a wide class of DLs, such as $\mathcal{FL}$ [Levesque and Brachman, 1985]; furthermore, it also establishes the non-learnability of many plausible extensions of C-Classic.

Unfortunately, the same method cannot be used to obtain positive results, as the converse of the proposition is false: there are some languages for which subsumption is tractable that are hard to pac-learn. For example, the DL containing only primitives and the `AND`, `ALL` and `SAME-AS` constructors is not pac-learnable, even though a tractable subsumption algorithm for this language exists [Cohen and Hirsh, 1992b]. This negative result can be easily extended to the DLs Classic1 and Classic2, which include the `SAME-AS` constructor.

```
Function LCS(v_1, v_2):
begin
    let v_LCS be the root of the tree to output
    let the label of v_LCS be (dom, prim, mn, mx) where
        (dom_1, prim_1, mn_1, mx_1) is the label of v_1
        (dom_2, prim_2, mn_2, mx_2) is the label of v_2
        dom = dom_1 ∪ dom_2
        prim = prim_1 ∩ prim_2
        mn = min(mn_1, mn_2)
        mx = max(mx_1, mx_2)
    for each edge from v_1 to w_1 with
     label (r, least_1, most_1, fillers_1)
        if there is an edge from v_2 to w_2
         with label (r, least_2, most_2, fillers_2)
        then
            let least = min(least_1, least_2)
            let most = max(most_1, most_2)
            let fills = fillers_1 ∩ fillers_2
            let w = LCS(w_1, w_2)
            construct an edge from v with w
             with label (r, least, most, fillers)
        endif
    endfor
end LCS function
```

Figure 2: An LCS Algorithm for C-CLASSIC

The principle technique for obtaining a positive result for a language $\mathcal{L}$ is to find a pac-learning algorithm for $\mathcal{L}$. An operation that is frequently useful in learning is finding a least general concept that is consistent with a set of positive examples; thus, we have also studied the complexity of computing *least common subsumers* (LCS) of a set of descriptions. An LCS of a set of descriptions $D_1,\ldots,D_m \in \mathcal{L}$ is simply a most specific description (in the infinite space of all possible descriptions in $\mathcal{L}$) that subsumes all of the D_i's. An LCS can also be thought of as the dual of the intersection (`AND`) operator, or as encoding the largest expressible set of commonalities between a set of descriptions.

A fairly general method for implementing LCS algorithms is described by Cohen, Borgida and Hirsh [Cohen *et al.*, 1992]. This method can be used to derive an LCS algorithm for any description logic which uses "structural subsumption": this class includes C-CLASSIC, but not full CLASSIC2. An LCS algorithm for C-CLASSIC description trees is shown in Figure 2.[2] This algorithm produces a unique LCS for any set of descriptions, and is tractable in the following sense: if $D_1,\ldots, D_m$ are all C-CLASSIC descriptions of size less than or equal to n_e, their LCS can be computed in time polynomial in m and n_e. We omit proofs of the correctness and tractability of the LCS procedure, and of the uniqueness of the LCS for C-CLASSIC [Cohen and Hirsh, 1992a].

[2] In the code, v_1 and v_2 are roots of two description trees. We also adopt the conventions that $S \cup \texttt{UNIV} = \texttt{UNIV} \cup S = \texttt{UNIV}$ for any set S, and $\max(n, \texttt{NOLIMIT}) = \min(n, \texttt{NOLIMIT}) = \texttt{NOLIMIT}$ for any real number n.

3 C-CLASSIC IS PAC-LEARNABLE

Often it is true that any algorithm that always returns a small hypothesis in $\mathcal{L}$ that is consistent with the training examples will pac-learn $\mathcal{L}$; thus often, if the LCS of a set of examples can be tractably computed for a language $\mathcal{L}$, computing the LCS of all the positive examples is a pac-learning algorithm for $\mathcal{L}$. Unfortunately, this is not the case for C-CLASSIC. As a counterexample, consider the target concept `THING`, and a distribution that is uniform over the examples $(\texttt{ONE-OF I}_1)^+$, $(\texttt{ONE-OF I}_2)^+$, $\ldots(\texttt{ONE-OF I}_r)^+$ where the $\texttt{I}_j$'s are distinct individuals. The LCS of any m examples will be the description $(\texttt{ONE-OF I}_{j_1} \ldots \texttt{I}_{j_m})$; in other words, it will simply be a disjunction of the positive examples. It can easily be shown that this does not satisfy the requirements for pac-learning when $r >> m$.

This example suggests that to pac-learn C-CLASSIC one must avoid forming large `ONE-OF` expressions. The LCSLEARN algorithm is one way of doing this. The LCSLEARN algorithm takes two inputs: a set of positive examples S^+ and a set of negative examples S^-, all of which are normalized CLASSIC2 descriptions. The algorithm behaves as follows.

1. If there are no positive examples, return the empty description `NOTHING`. Otherwise, let H be the LCS of all of the positive examples, and let $l = 0$.
2. Let H_l be a copy of H in which every `ONE-OF` label in H that contains more than l individuals is deleted.
3. If H_l does not subsume any negative example e^- in S^-, then return H_l as the hypothesis of LCSLEARN. Otherwise, if $H_l = H$ then abort with failure. Otherwise, increment l and go to Step 2.

The main formal result of this paper is the following.[3]

Theorem 2 LCSLEARN *is a pac-learning algorithm for* C-CLASSIC*, with a sample complexity of no more*

[3] Note that this theorem assumes a size measure on C-CLASSIC concepts. We define the size of a description to be the size of the equivalent canonical description tree, and that the size of a description tree is the sum of the number of vertices, the number of edges, and the sum of the sizes of all the labels, where a label that is a symbol or a real number has size one, and a label that is a set S has size $|S|$.

than

$$m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_t) \equiv \max(\frac{8n_t + 4n_e}{\epsilon} \ln \frac{4n_t + 2n_e}{\sqrt{\delta}}, \frac{32n_e}{\epsilon} \ln \frac{26}{\epsilon}) \equiv O(\frac{n_t + n_e}{\epsilon} \ln \frac{n_t + n_e}{\sqrt{\delta}} + \frac{n_e}{\epsilon} \ln \frac{1}{\epsilon})$$

regardless of the number of primitive concepts and roles.

Proof: In the proof, we will analyze the behavior of an incremental version of LCSLearn called IncLCSLearn. This will allow us to use proof techniques from mistake-bounded learning [Littlestone, 1988] for part of the proof, thereby achieving a sample complexity independent of the number of roles and primitives.

IncLCSLearn examines the examples in S^+ and S^- in some randomly chosen order. After examining the i-th example x_i, IncLCSLearn generates a hypothesis H_i, which is defined to be the hypothesis that LCSLearn would output from a sample containing the first i examples $x_1, \ldots, x_i$. IncLCSLearn returns as its hypothesis the first H_i such that

Property 1. $i > \max(\frac{8}{\epsilon} \ln \frac{4}{\delta}, \frac{32n_e}{\epsilon} \ln \frac{26}{\epsilon})$

Property 2. IncLCSLearn has made no nonboundary errors (defined below) on the $m_j = \frac{2}{\epsilon} \ln \frac{4j^2}{\delta}$ previous examples, where j is the number of previous nonboundary mind changes (defined below) made by IncLCSLearn.

We will show that IncLCSLearn is a pac-learner with the stated sample complexity. Since IncLCSLearn's hypothesis is the same as LCSLearn would generate given a subset of the data, this implies that LCSLearn is also a pac-learner.

Define a *mind change* to be any occasion in which H_i differs from H_{i-1}. A *boundary mind change* is one in which (a) H_i and H_{i-1} have the same number of edges and vertices, (b) for each vertex only the *mn* and *mx* labels change and (c) for each edge only the *least* and *most* labels change. A *nonboundary mind change* is any other sort of mind change. Also define a *prediction error* for an incremental learner to be any occasion in which the i-th hypothesis H_i misclassifies the $(i+1)$-th example x_{i+1}. A *boundary error* is an error in which a positive example x_i is not subsumed by H_i, but it would have been subsumed if every *least* label in H_i were replaced by zero, and every *most*, *mn* and *mx* label by `NOLIMIT`. A *nonboundary error* is any other sort of error.

The proof begins with two lemmas, the proofs of which are omitted. (The first proof uses a standard Chernoff bound argument; the second is a straightforward application of the main result of Blumer *et. al* [1989], and a bound of $2d$ on the VC-dimension of the language d-$\mathcal{L}_{\mathrm{RECT}}$.)

Lemma 1 *Let α be some type of prediction error (e.g. a nonboundary error) and let H_i be some hypothesis of an incremental learner that was formed from the examples $x_1, \ldots, x_i$ and that makes no prediction errors of type α on the subsequent examples $x_{i+1}, \ldots, x_{i+m_j}$, where $m_j = \frac{1}{\epsilon} \ln \frac{2j^2}{\delta}$ and j is the number of previous hypotheses of the incremental learner* IncLearn *that are different with respect to the type-α errors that they could make. Then with probability at least $1-\delta$, H_i will have probability less than ϵ of making a type-α prediction error on a randomly chosen example.*

Lemma 2 *Let d-$\mathcal{L}_{\mathrm{RECT}}$ be the set of d-dimensional rectangles whose boundaries are specified by real numbers, and define rectangle R_1 to subsume rectangle R_2 iff the points contained in R_1 are a superset of the points contained in R_2. Then d-$\mathcal{L}_{\mathrm{RECT}}$ is pac-learnable with a sample complexity of*

$$\max(\frac{4}{\epsilon} \ln \frac{2}{\delta}, \frac{16d}{\epsilon} \ln \frac{13}{\epsilon})$$

by any learning algorithm that outputs a rectangle consistent with all of the examples.

With these tools in hand, we can now prove the theorem. By Lemma 1, any hypothesis returned by IncLCSLearn will with confidence $\frac{\delta}{2}$ have probability less than $\frac{\epsilon}{2}$ of making a nonboundary error on a new random example. Now consider boundary errors. Finding the right values for the *least*, *most*, *mn*, and *mx* labels in a tree of size n is equivalent to finding an accurate hypothesis in $(v+e)$-$\mathcal{L}_{\mathrm{RECT}}$, where v is the number of vertices in the tree and e is the number of edges. Since for any hypothesis tree H_i, $v + e < n_e$, by Lemma 2 any hypothesis that is consistent with

$$\max(\frac{8}{\epsilon} \ln \frac{4}{\delta}, \frac{32n_e}{\epsilon} \ln \frac{26}{\epsilon}) \quad (1)$$

examples will with confidence $\frac{\delta}{2}$ have probability less than $\frac{\epsilon}{2}$ of making a boundary error on a new random example. Thus the hypothesis of IncLCSLearn will with confidence δ have error less than ϵ.

It remains to bound the number of examples required to satisfy Property 2. The worst case would be to make no nonboundary mind changes on the first $m_1 - 1$ examples, then after a nonboundary mind change on the m_1-th example to make no nonboundary mind changes on the next $m_2 - 1$ examples, and so on. Thus the number of examples required to satisfy Property 2 can be bounded by $\sum_j m_j$. Notice first that the number of nonboundary mind changes can be bounded as follows:

- The hypothesized bound l on *dom* labels can be incremented at most n_t times.

- A set representing a *dom* label can be increased in size to at most n_t, or can be changed from a set to `NOLIMIT`. Since *dom* labels are initially non-empty, they can be changed at most n_t times (independently of changes to the bound l on *dom* labels.)
- The total number of times that a *prim* or *fillers* label is removed or that an edge or vertex is be removed from the tree is bounded by n_e.

Thus we see that the number of nonboundary errors for IncLCSLearn is bounded by $2n_t + n_e$. The sum of the m_j's can now be bounded as follows:

$$\sum_{j=1}^{2n_t+n_e} m_j \leq (2n_t+n_e)m_{(2n_t+n_e)}$$
$$= \frac{8n_t+4n_e}{\epsilon} \ln \frac{4n_t+2n_e}{\sqrt{\delta}}$$

By combining this with Equation 1 we obtain the sample complexity given in the statement of the theorem. ■

This result extends the previous results of Cohen and Hirsh [1992b] to include a larger set of constructors. This result can be also extended to allow a limited use of the `SAME-AS` constructor by imposing restrictions on the use of `SAME-AS` analogous to those described by Cohen and Hirsh.

4 EXPERIMENTAL RESULTS

In the formal model described above, examples are assumed to be descriptions, and an example is marked as positive if it is subsumed by the target concept. While convenient for formal analysis, this assumption is not always appropriate; in many cases, it is desirable to use instead *individuals* as examples. The formal results also give only a loose polynomial bound on learning speed.

Thus an experimental investigation of the behavior of LCSLearn is desirable. In the remainder of this section, we will describe a simple means of extending the LCSLearn algorithm to learn from individuals, and present some experimental results with the extended algorithm.

4.1 LEARNING FROM INDIVIDUALS

A straightforward way of adapting LCSLearn to learn from individuals is to provide a preprocessor that *abstracts* an individual I by constructing a very specific description d_I that subsumes I. If one can guarantee that when an individual is an instance of C its abstraction will be subsumed by C, then many of the desirable formal properties of LCSLearn are preserved. Suppose, for example, that d_I is always the least general concept that contains I in some sublanguage $\mathcal{L}_0$ of C-Classic. Then applying LCSLearn to abstracted training examples is a pac-learning algorithm for $\mathcal{L}_0$.

We have experimented with a number of methods to abstract individuals. The simplest of these abstraction methods finds the least general description d_I that (a) contains no `SAME-AS` restrictions and (b) contains at most k levels of nesting of the `ALL` constructor, for value of k provided by the user. It is easy to show that this least general concept is unique, and can be found in time polynomial in the size of the knowledge base (but exponential in k). We used this strategy with $k = 3$ in the experiments with the Imacs2 knowledge base (see below), and $k = 5$ for all of the other experiments in this paper.

4.2 RECONSTRUCTING CONCEPT HIERARCHIES

In our first set of experiments, we used LCSLearn to reconstruct known concept hierarchies from examples. Each concept c in the hierarchy was made a target concept for LCSLearn, with the instances of c serving as positive examples, and non-instances of c serving as negative examples.

By reconstructing concept hierarchies from a variety of knowledge bases we were able to test LCSLearn on a large number of naturally occurring concepts—almost 1000 all told. Some of these concepts were simple, but others were quite complex. The largest concept in our benchmark suite has a description more than 10,000 symbols long; for one of the knowledge bases (Prose1) the *average* description size was more than 2000 symbols.

We evaluated the learning algorithm on each knowledge base in two ways. First, we measured the fraction of the concepts in each knowledge base for which the hypothesis of LCSLearn was equivalent to the true target concept. Somewhat surprisingly, in several of the domains a significant fraction of the hypotheses met this stringent test. Second, we estimated the error rate of each hypothesis using the statistical technique of cross-validation[4] [Weiss and Kulkowski, 1990].

Table 2 contains the results of this experiment. The Wines knowledge base is the one distributed with Classic2. The Imacs1 knowledge base is the one used as a running example by Brachman *et. al* [1992], and Imacs2 is a small knowledge base used to test a real-world application of the system of [Brachman *et al.*, 1992]. Prose1 and Prose2 are knowledge bases used for different hardware configuration tasks [Wright *et al.*, 1993]. KRK, Loan, and Kluster are knowledge bases

[4] For problems with 100 or more examples, we used 20 partitions. For problems with less than 100 examples, we used "leave-one-out" cross-validation.

Table 2: Using individuals to reconstruct hierarchies

KB	#Concepts	#Individuals	Equivalent to Target	Error rate	
				LCSLearn	Default
Wines	134	177	37/134	0.49%	3.5%
Imacs1	9	2564	2/9	0.063%	11.1%
Imacs2	74	512	19/74	0.31%	2.1%
Prose1	301	293	1/301	0.031%	0.34%
Prose2	398	202	1/398	0.092%	0.60%
KRK	16	1049	5/16	0.089%	6.4%
Loan	22	1013	3/22	0.031%	15.3%
Kluster	13	16	2/13	7.7%	17.8%
Total	967		70/967		
Average	121.0	728.25		1.15%	6.87%

used to compare LCSLearn with other work in learning first-order concepts. KRK classifies king-rook-king chess positions as legal or illegal [Quinlan, 1990; Pazzani and Kibler, 1992], Loan determines if payment can be deferred on a student loan [Pazzani and Brunk, 1991], and Kluster encodes a pharmacological domain [Kietz and Morik, 1991]. KRK and Loan were translated from Prolog, and Kluster was translated from BACK [Peltason *et al.*, 1991].

To summarize the results, LCSLearn finds very accurate hypotheses in all of the domains except for Kluster, which has few individuals and hence affords little training data for the learning algorithm.

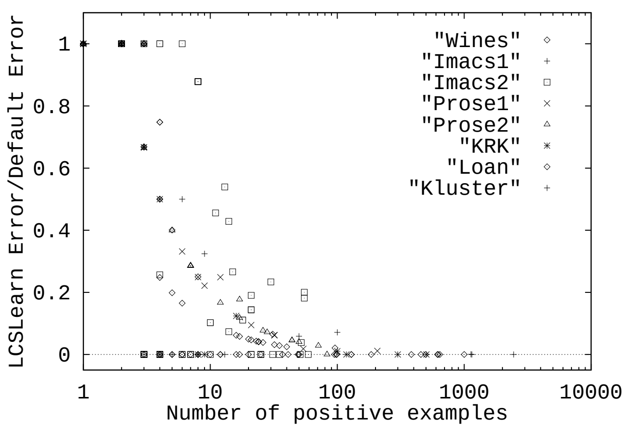
Some of the knowledge bases include many concepts with few instances: for such concepts hypothesizing the empty description **NOTHING** would also give a low error rate. Thus we also give for each knowledge base the error rate of the *default rule*.[5] LCSLearn outperforms the default rule on all of the knowledge bases, often having an average error rate more than an order of magnitude lower.

4.3 ANALYSIS OF RESULTS

Most implemented learning systems are tested on at most a few dozen learning problems. In the experiments above, however, we have evaluated LCSLearn on several hundred benchmarks. This provides sufficient data to make some general statements about its performance.

In Figure 3, we have plotted one point for each benchmark problem. The x coordinate of each point is the log of the number of positive examples,[6] and the y coordinate is the cross validated error rate of LCSLearn divided by the default error rate; thus $y > 1$ indicates performance worse that the default rule, and $y \approx 0$ indicates performance much better than the default rule. This plot shows in general, the performance of LCSLearn relative to the default rule improves quickly when more positive examples are available. This is unsurprising, when one considers that LCSLearn derives most of its information from the positive examples. Also (on these benchmarks) LCSLearn never performs worse than the default rule, and LCSLearn outperforms the default rule whenever there are more than a handful of positive examples.

Figure 3: Further analysis of LCSLearn



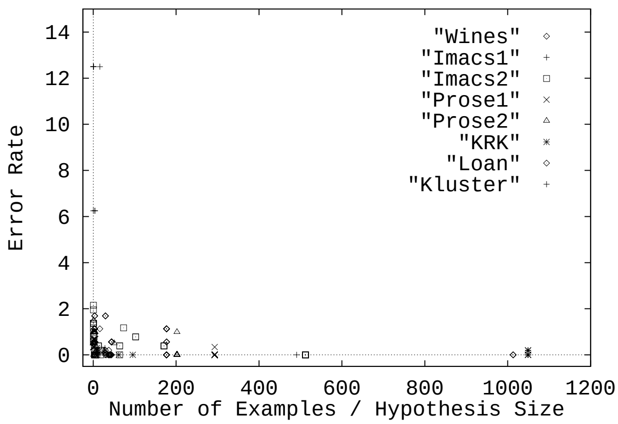
One might also expect that when LCSLearn outputs a small hypothesis that is consistent with many examples, that hypothesis is likely to have low error. Figure 4 plots the ratio of number of examples m to hypothesis size n_h, on the x axis, against error rate, on the y axis.[7] Most of the points are clustered near the origin, indicating that for most of the benchmarks

[5] The default rule simply predicts "positive" if more than half of the training examples are positive, and predicts "negative" otherwise.

[6] Logs are used because of the large variation in the amount of training data.

[7] For readability, we show only the part of this plot clos-

Figure 4: Effect of number of examples on error rate



both the error rate e and the ratio $\frac{m}{n_h}$ are low. The outlying points lie entirely near either the x axis or y axis, showing that error rate e is high only when $\frac{m}{n_h}$ is very low, and conversely that whenever $\frac{m}{n_h}$ high the error rate e is very low.

4.4 RELATIONSHIP TO KLUSTER

This application of LCSLearn is somewhat similar to an earlier learning system called Kluster [Kietz and Morik, 1991]. Unfortunately, Kluster has not been systematically evaluated over a range of domains, which makes quantitative comparison of Kluster and LCSLearn difficult. LCSLearn's performance on the benchmark knowledge based described by Kietz and Morik is given in Table 2; in this section we will comment on the qualitative similarities and differences between LCSLearn and the learning component of Kluster.

As in the experiments above, Kluster's starting point is a set of individuals which are linked by roles and classified as to the primitive concepts to which they belong.[8] Kluster first heuristically partitions the individuals into disjoint classes, and then learns a description for each class expressed in a subset of the BACK description logic, using as examples the instances and non-instances of the constructed class. Kluster's learning algorithm first uses a sound LCS-like method to learn a description in a sublanguage of descriptions of the form

$$
\begin{aligned}
&(\texttt{AND}\ (\texttt{ALL}\ r_1\ (\texttt{AND}\ p_{1,1}\ \ldots p_{1,n_1}))\\
&\quad (\texttt{AT-LEAST}\ l_1\ r_1)\ (\texttt{AT-MOST}\ m_1\ r_1)\\
&\quad \vdots\\
&\quad (\texttt{ALL}\ r_k\ (\texttt{AND}\ p_{k,1}\ \ldots p_{k,n_k}))\\
&\quad (\texttt{AT-LEAST}\ l_k\ r_k)\ (\texttt{AT-MOST}\ m_k\ r_k))
\end{aligned}
$$

where the r_i's are named roles, the l_i's and m_i's are integers, and the $p_{i,j}$'s are named concepts. To circumvent the restrictiveness of this language, heuristic techniques are then used to introduce new named concepts and named roles. Finally, heuristic techniques are again used to generalize the resulting descriptions.

In contrast, LCSLearn uses only one heuristic step—the abstraction of individuals—and learns descriptions in Classic2, a language more expressive than the sublanguage above, and incomparable to the full language that is learned by the Kluster technique. Classic2 is more expressive than the subset of BACK used in Kluster in that it includes the `MIN`, `MAX`, `FILLS`, `TEST`, `SAME-AS`, and `ONE-OF` constructors, and allows `ALL` restrictions to be nested; however, Classic2 does not allow defined roles, and hence it is also in some respects less expressive. To circumvent this limitation it was necessary in the experiments above to add to the original ontology the two roles defined by Kluster.

5 LEARNING DISJUNCTIONS

Because Classic contains only limited disjunction (in the `ONE-OF` constructor) many target concepts of practical interest can not be expressed by a single Classic2 description. One way to relax this limitation is to consider algorithms that learn a disjunction of descriptions, rather than a single Classic concept; in other words, to learn a target concept $c \equiv d_1 \vee d_2 \vee \ldots \vee d_n$ where each d_i is a Classic2 description.

Learning disjunctions of Classic concepts is somewhat analogous to the problem of "inductive logic programming" (ILP) [Quinlan, 1990; Muggleton and Feng, 1992]. In ILP the target concept is usually assumed to be a single Prolog predicate that is defined by a set of Prolog clauses; such a concept can often be viewed as a disjunction of the sets defined by each clause. Thus one natural approach to learning disjunctions of Classic descriptions is to adapt the techniques used in ILP to learn multi-clause Prolog predicates.

One well-known ILP method for learning multiple clauses is the GOLEM algorithm [Muggleton and Feng, 1992], which is also based on computing least common generalizations. The basic idea behind this algorithm is to use LCS to implement a specific-to-general greedy search for descriptions that cover many positive examples and no negative examples. In GOLEM, these descriptions are then further generalized by a process called *reduction*, and finally disjoined

est to the origin.

[8]As Classic allows arbitrary assertions to be made about individuals, more information about individuals can be available for LCSLearn; however in most of the domains above this is not the case.

```
LCSLearnDisj(S+,S-)
  n ← 0
  while S+ is nonempty do
    n ← n + 1
    Seed ← FindSeed(S+,S-)
    d_n ← Generalize(Seed,S+,S-)
    remove from S+ all examples in d_n
  endwhile
  return d_1 ∨ ... ∨ d_n

FindSeed(S+,S-)
  R ← c_1 random elements from S+
  PAIRS ← {LCS1(r_i, r_j) : r_i, r_j ∈ R}
  discard from PAIRS all descriptions that
    contain any negative examples e- ∈ S-
  return the p ∈ PAIRS that contains
    the most positive examples e+ ∈ S+

Generalize(Seed,S+,S-)
  repeat
    R ← c_2 random elements from S+
        that are not covered by Seed
    GENS ← {LCS1(Seed, r_i) : r ∈ R}
    discard from GENS all descriptions that
      contain any negative examples e- ∈ S-
    if GENS is nonempty then
      Seed ← the g ∈ GENS that contains
        the most positive examples e+ ∈ S+
    endif
  until GENS is empty
  return Reduce(Seed,S-)

LCS1(d_1,d_2)
  return a copy of LCS(d_1,d_2) with
    all ONE-OF restrictions removed
```

Figure 5: The LCSLearnDisj learning algorithm

```
Reduce(D,N)
  n ← 0
  while N is nonempty do
    n ← n + 1
    f_n ← the f ∈ Factors(D) maximizing |{x ∈ N : x ∉ f}|
    N ← N − {x ∈ N : x ∉ f_n}
  endwhile
  return f_1 ... f_n

Factors(D)
  if D = (AND D_1 ... D_k) then
    return ∪_{i=1}^{k} Factors(D_i)
  elseif D = (ALL r D) then
    return {(ALL r f) : f is a factor of D}
  elseif D = (FILLS I_1 ... I_k) then
    return {(FILLS I_1), ..., (FILLS I_k)}
  else return D
```

Figure 6: Reducing C-Classic Descriptions

to obtain a hypothesis.

Figure 5 gives a brief overview of the algorithm as adapted to Classic.[9] To *reduce* a description D, we first "factor" it into a set of simpler descriptions $\{f_1, \ldots, f_n\}$ such that the intersection of the f_i's is equivalent to D. We then use a greedy set-covering approach to find a small subset of the factors of D that, when conjoined, are consistent with the negative data. The details of the reduction algorithm are given in Figure 6.

Three of the knowledge bases above are useful test cases for this learning algorithm. The KRK and Loan knowledge bases, being adaptations of ILP problems, naturally fall into this category. We ran LCSLearnDisj on these benchmarks and also on some obvious variants of the KRK problem, shown in the table as KBK and KQK.[10]

The third test case is the Wines knowledge base, which contains a set of rules that recommend which wines to serve with which foods. From these rules we derived a number of learning problems. First, we derived 12 disjunctive concepts defining the foods that are acceptable with 12 different types of wines: for example, the disjunctive concept `Color-Red-Food` contains those foods that can be served with red wine. The training examples for these concepts are just the 33 food individuals in the knowledge base. We also derived a single disjunctive concept containing exactly the $(wine, food)$ pairs deemed acceptable by the Wine rules. We generated a dataset for the "acceptable pair" concept by choosing a set of $(wine, food)$ pairs, and then classifying these pairs as acceptable or unacceptable using the rules from the Wines knowledge base; the generated dataset contains all acceptable pairs and a random sample of 10% of the unacceptable pairs.

Table 3 summarizes these experiments; for convenience, the 12 smaller wine problems are also summarized in a single line labeled "Acceptable-Food". LCSLearnDisj is the name given to our learning algorithm; the other points of comparison that we use are Grendel2, a recent version of the ILP learning system Grendel [Cohen, 1992; Cohen, 1993], and the default error rate.[11]

[9] In the experiments we used $c_1 = 5$ and $c_2 = 20$. Note that the limited disjunction provided by `ONE-OF` is no longer needed, as a more general mechanism for disjunction is being provided, hence the use of LCS1 rather than LCS.

[10] The white rook is replaced by a white bishop in KBK-Illegal and by a queen in KQK-Illegal.

[11] Results for Grendel are in each case the best results obtained among a variety of different expressible biases [Cohen, 1993]. Grendel was not applied to the Wines problems

Table 3: Learning disjunctions of CLASSIC concepts

KB	#Examples	Error rate		
		LCSLEARNDISJ	Default	Grendel2
KRK	100	2.0%	38.0%	3.0%
KBK	100	2.0%	36.0%	54.0%
KQK	100	3.0%	44.0%	55.0%
Loan	100	13.0%	35.0%	2.0%
Acceptable-Pair	320	5.3%	42.5%	—
Acceptable-Foods:				—
Color-White-Food	33	12.2%	33.3%	—
Color-Rose-Food	33	0.0%	6.1%	—
Color-Red-Food	33	6.1%	39.4%	—
Body-Light-Food	33	3.0%	9.1%	—
Body-Medium-Food	33	12.2%	45.5%	—
Body-Full-Food	33	6.1%	36.4%	—
Flavor-Delicate-Food	33	6.1%	21.2%	—
Flavor-Moderate-Food	33	9.1%	39.4%	—
Flavor-Strong-Food	33	9.1%	42.4%	—
Sugar-Sweet-Food	33	3.0%	27.3%	—
Sugar-OffDry-Food	33	3.0%	3.0%	—
Sugar-Dry-Food	33	6.1%	30.3%	—
Average Acceptable-Food	33	6.3%	27.8%	—

The results of Table 3 show that LCSLEARNDISJ obtains good results, and suggests that it is competitive with existing first-order learning methods. However, it should be noted that both LCSLEARNDISJ and ILP systems like Grendel are sensitive to the way examples are represented, and ILP systems and LCSLEARNDISJ necessarily use different representations.

To illustrate the differences in representation, we will briefly discuss our translation of the KRK learning problem. In this domain, the task is to classify king-rook-king chess positions with white to move as legal or illegal. In the formulation of this problem used by Grendel2, a position is represented by six numbers encoding the rank and file position of each of the tree pieces. Illegal positions are recognized by checking arithmetic relationships between pairs and triples of numbers. An an example, the Prolog clause below states that a position is illegal if the white rook and black king are on the same file and the white king does not block the white rook from attacking the black king. (The variables WKR, WKF, WRR, WRF, BKR, and BKF stand for the rank and file of the white king, the rank and file of the white rook, and the rank and file of the black king respectively.)

```
illegal(WKR,WKF,WRR,WRF,BKR,BKF) ←
    WRF=BKF,
    ( WKF=BKF, not between(WRR,WKR,BKR)
    ; not WKF=BKF ).
```

In the C-CLASSIC formulation of the problem, a position has the attributes `white-king`, `white-rook`, and `black-king`, each of which is filled by an individual that must be a `piece`; a `piece` has the attribute `location`, which must be filled by a `square`; and a `square` has the attributes `rank` and `file` and a role `content`, which must be filled by one or more `pieces`. Finally, to encode the spatial relationships among pieces, every piece also has a number of attributes with names like `to-white-rook` and `to-black-king` that are filled by `vector` individuals. A `vector` individual is related to all the `squares` between its two endpoints via the role `between`, and also has a `direction` attribute. The filler of the *direction* attribute is one of the individuals `n`, `s`, `e`, `w`, `ne`, `se`, `nw`, or `sw`, and these individuals are organized in a taxonomy that includes concepts like `diagonal-direction` and `file-direction`.

Using this ontology, the Prolog clause above can be translated as the following C-CLASSIC concept.

```
(ALL white-rook
    (ALL to-black-king
        (AND
         (ALL direction file-dir)
         (ALL between (AT-MOST 0 content)))))
```

Both the ILP and C-CLASSIC representations are natural given the choice of languages; however, as the example shows, the representations are also both quite different, and have different strengths and weaknesses. The C-CLASSIC representation makes it possible to concisely describe certain geometric patterns that are difficult to express in the Prolog representation, such as an unobstructed line of attack along a diagonal.

because they are not represented in an ontology conducive to an ILP representation.

The Prolog representation, on the other hand, allows a very compact representation of a position.

To summarize, the differences in the languages greatly complicate comparisons between ILP techniques and learning methods based on description logics: not only does the background knowledge used by the two systems differ, but the representation of the examples themselves is also different. (This is in marked contrast to comparisons among different learning systems based on propositional logic, in which the same representations are typically used for examples.) However, we believe that the experiments above do clearly indicate that LCSLearn can be competitive with ILP methods, given an appropriate ontology.

6 CONCLUDING REMARKS

The description logic C-Classic has been used in a number of practical systems. In this paper, we have presented a formal result showing that the description logic C-Classic is pac-learnable. The learning algorithm LCSLearn suggested by this formal result learns descriptions in the C-Classic description logic from examples which are themselves descriptions.

Additionally, we have presented an experimental evaluation of two extensions to the algorithm: one that learns from examples that are individuals (by simply converting each example individual to a very specific concept that includes that individual) and a second that learns disjunctions of descriptions from individuals. Extensive experiments with LCSLearn using several hundred target concepts from a number of domains support the claim that the learning algorithm reliably learns complex natural concepts, in addition to having behavior that is formally well understood.

Similar experiments with the extension of LCSLearn that learns disjunctions suggest that it is competitive with existing techniques for learning first-order concepts from examples. This suggests that learning systems based on description logics may prove to be a useful complement to those based on logic programs as a representation language.

Acknowledgments

Thanks are due to Lori Alperin Resnick, for writing the LCS routine for Classic2 and her invaluable help with Classic2, and Deborah McGuinness, for helping me assemble the set of benchmark knowledge bases. Alon Levy also provided valuable comments on a draft of this paper, as did the anonymous reviewers for KR94.

References

(Beck *et al.*, 1989) H. Beck, H. Gala, and S. Navathe. Classification as a query processing technique in the CANDIDE semantic model. In *Proceedings of the Data Engineering Conference*, pages 572–581, Los Angeles, California, 1989.

(Blumer *et al.*, 1989) Anselm Blumer, Andrezj Ehrenfeucht, David Haussler, and Manfred Warmuth. Classifying learnable concepts with the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, 1989.

(Borgida and Patel-Schneider, 1992) A. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the CLASSIC description logic. AT&T Technical Memorandum. Available from the authors on request, 1992.

(Borgida *et al.*, 1989) A. Borgida, R. J. Brachman, D. L. McGuinness, and L. Resnick. CLASSIC: A structural data model for objects. In *Proceedings of SIGMOD-89*, Portland, Oregon, 1989.

(Brachman *et al.*, 1992) R. J. Brachman, P. G. Selfridge, L. G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D. L. McGuinness, and L. A. Resnick. Knowledge representation support for data archaeology. In *Proceedings of the ISMM International Conference on Information and Knowledge Management (CIKM-92)*, pages 457–464, Baltimore, MD, 1992.

(Brachman, 1990) Ron J. Brachman. Living with CLASSIC: when and how to use a KL-ONE-like language. In J. Sowa, editor, *Formal aspects of Semantic Networks*. Morgan Kaufmann, 1990.

(Cohen and Hirsh, 1992a) W. Cohen and H. Hirsh. Learnability of the CLASSIC 1.0 knowledge representation language. AT&T Bell Labs Technical Memorandum. Available from the author on request, 1992.

(Cohen and Hirsh, 1992b) William W. Cohen and Haym Hirsh. Learnability of description logics. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, Pittsburgh, Pennsylvania, 1992. ACM Press.

(Cohen *et al.*, 1992) 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*, San Jose, California, 1992. MIT Press.

(Cohen, 1992) William W. Cohen. Compiling knowledge into an explicit bias. In *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland, 1992. Morgan Kaufmann.

(Cohen, 1993) William W. Cohen. Rapid prototyping of ILP systems using explicit bias. In *Proceedings of the 1993 IJCAI Workshop on Inductive Logic Programming*, Chambery, France, 1993.

(Devanbu *et al.*, 1991) P. Devanbu, R. J. Brachman, P. Selfridge, and B. Ballard. LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 35(5), May 1991.

(Dietterich *et al.*, 1982) Thomas Glen Dietterich, Bob London, Kenneth Clarkson, and Geoff Dromey. Learning and inductive inference. In Paul Cohen and Edward A. Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume III*. William Kaufmann, Los Altos, CA, 1982.

(Haussler, 1989) David Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1), 1989.

(Kietz and Morik, 1991) Jorg-Uwe Kietz and Katharina Morik. Constructive induction of background knowledge. In *Proceedings of the Workshop on Evaluating and Changing Representation in Machine Learning (at the 12th International Joint Conference on Artificial Intelligence)*, Sydney, Australia, 1991. Morgan Kaufmann.

(Levesque and Brachman, 1985) Hector Levesque and Ronald Brachman. A fundamental tradeoff in knowledge representation and reasoning (revised version). In *Readings in Knowledge Representation*. Morgan Kaufmann, 1985.

(Littlestone, 1988) Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

(MacGregor, 1991) R. M. MacGregor. The evolving technology of classification-based knowledge representation systems. In John Sowa, editor, *Principles of semantic networks: explorations in the representation of knowledge*. Morgan Kaufmann, 1991.

(Mays *et al.*, 1987) E. Mays, C. Apte, J. Griesmer, and J. Kastner. Organizing knowledge in a complex financial domain. *IEEE Expert*, pages 61–70, Fall 1987.

(Muggleton and Feng, 1992) Stephen Muggleton and Cao Feng. Efficient induction of logic programs. In *Inductive Logic Programming*. Academic Press, 1992.

(Pazzani and Brunk, 1991) Michael Pazzani and Clifford Brunk. Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning. *Knowledge Acquisition*, 3:157–173, 1991.

(Pazzani and Kibler, 1992) Michael Pazzani and Dennis Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1), 1992.

(Peltason *et al.*, 1991) C. Peltason, A. Schmiedel, C. Kindermann, and J. Quantz. The BACK system revisted. Technical Report KIT-REPORT 75, Technical University of Berlin, 1991.

(Quinlan, 1990) J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.

(Schapire, 1990) Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2), 1990.

(Valiant, 1984) L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), November 1984.

(Weiss and Kulkowski, 1990) Sholom Weiss and Casmir Kulkowski. *Computer Systems that Learn*. Morgan Kaufmann, 1990.

(Woods and Schmolze, 1992) W. A. Woods and J. G. Schmolze. The KL-ONE family. *Computers And Mathematics With Applications*, 23(2-5), March 1992.

(Wright *et al.*, 1993) Jon Wright, Elia Weixelbaum, Gregg Vesonder, Karen Brown, Spephen Palmer, Jay Berman, and Harry Moore. A knowledge-based configurator that supports sales engineering and manufacturing and AT&T network systems. *AI Magazine*, 14:69–80, 1993.