



# CONCEPT LEARNING IN THE $\mathcal{ALC}$ DESCRIPTION LOGIC

Jens Lehmann<sup>1</sup> and Pascal Hitzler<sup>2</sup>

<sup>1</sup> Department of Computer Science, Universität Leipzig, Germany – lehmann@informatik.uni-leipzig.de  
<sup>2</sup> AIFB Institute, Universität Karlsruhe, Germany – hitzler@aifb.uni-karlsruhe.de



## Description

### Motivation:

- ▶ The ontology language **OWL DL** based on **Description Logics** (DLs) is **widely used** in many application areas
- ▶ **Creating ontologies** is still a **burdensome** task (knowledge engineer and domain expert needed)
- ▶ The **application of ILP methods** to Description Logics has **not** been fully researched yet – both theoretical and practical

### Solution Approach:

- ▶ Transfer ILP methodology, in particular **refinement operators**, to DLs
- ▶ **Provide solid theoretical foundations** for using refinement operators
- ▶ Create an algorithm for **learning concepts from positive and negative examples** to enable creation, extension, and analysis of OWL ontologies
- ▶ System is provided for the  $\mathcal{ALC}$  description logic but **extensible** to more expressive languages
- ▶ Implemented system **DL-Learner**: top-down algorithm, redundancy checks, handles infinite operator, DIG interface for reasoner communication, ...

## Theoretical Foundations

- ▶ We researched the **properties** (completeness, properness, redundancy, finiteness, minimality) of **refinement operators**
- ▶ **Which properties can be combined?**
- ▶ We obtained general **results for any sufficiently expressive description language  $\mathcal{L}$**  (i.e.  $\mathcal{L}$  allows to express  $\top$ ,  $\perp$ , conjunction, disjunction, universal quantification, and existential quantification)

### Property Theorem

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

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

## Operator

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

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

(abbreviated representation: see the paper for definitions of  $\text{nb}_{\perp}$ ,  $\text{nb}_{\top}$ , and  $M$ )

**Result:** The closure of  $\rho_{\perp}$  is a **complete** and **proper**  $\mathcal{ALC}$  downward refinement operator.

## Algorithm

**Input:** *horizExpFactor* in ]0,1]

*ST* (search tree) is set to the tree consisting only of the root node ( $\top, 0, q(\top)$ , *false*)

*minHorizExp* = 0

**while** *ST* does not contain a correct concept **do**

  choose  $N = (C, n, q, b)$  with highest fitness in *ST*

  expand  $N$  up to length  $n + 1$ , i.e. :

**begin**

    add all nodes  $(D, n, -, \text{checkRed}(ST, D))$  with  $D \in \text{trans}(\rho_{\perp}^{cl}(C))$

    and  $|D| = n + 1$  as children of  $N$

    evaluate created non-redundant nodes

    change  $N$  to  $(C, n + 1, q, b)$

**end**

*minHorizExp* =  $\max(\text{minHorizExp}, \lceil \text{horizExpFactor} * (n + 1) \rceil)$

**while** there are nodes with defined quality and *horiz. expansion smaller minHorizExp* **do**

    expand these nodes up to *minHorizExp*

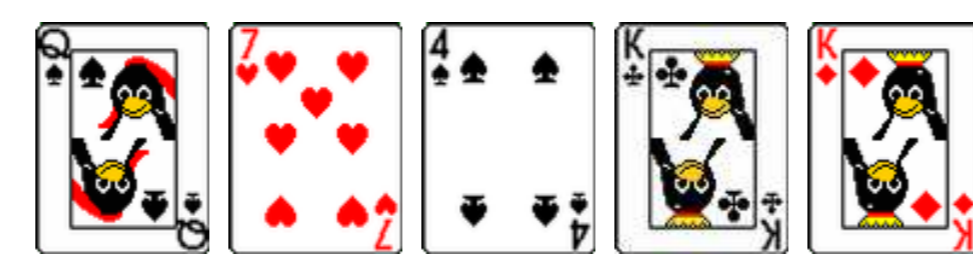
Return a correct concept in *ST*

## Evaluation

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

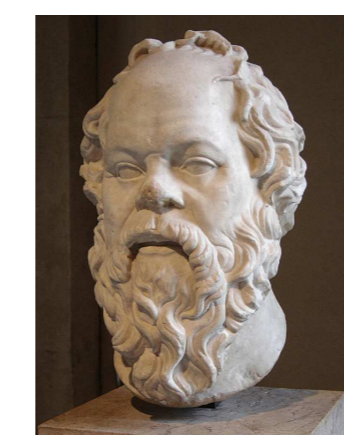
### Poker - Pair

$\exists \text{hasCard}.\exists \text{sameRank}.\top$



### Poker - Straight

$\exists \text{hasCard}.\exists \text{nextRank}.\exists \text{nextRank}.$

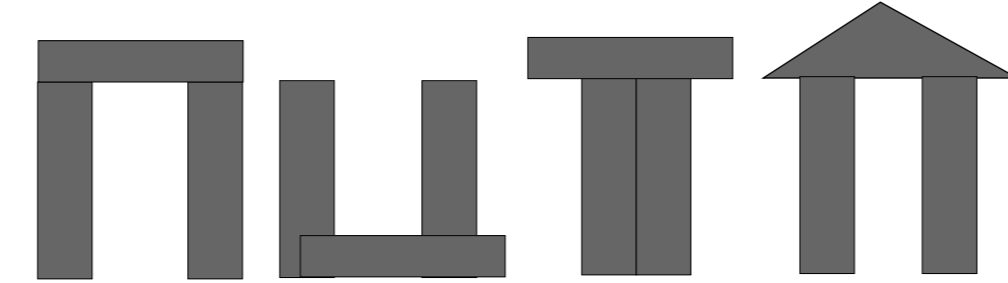


### Moral Reasoner

simple:  $\text{Guilty} \sqsubseteq \text{Blameworthy} \sqcup \text{Vicarious.blame}$

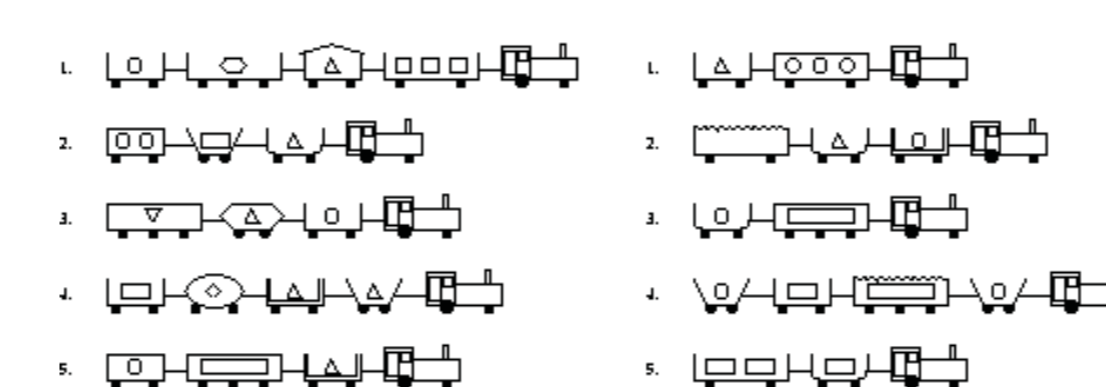
complex:  $\text{Guilty} \sqsubseteq \neg \text{Justified} \sqcap (\text{Vicarious}$

$\sqcup (\text{Negligent} \sqcap \text{Responsible}))$



### Arches

$\text{Arch} \sqsubseteq \exists \text{hasPillar}.\text{FreeStandingPillar} \sqcap$   
 $\exists \text{leftOf}.\exists \text{supports}.\top$



### Trains

$\text{Left} \sqsubseteq \exists \text{hasCar}.\text{Closed} \sqcap \text{Short}$

## Contributions & Future Work

### Contributions to the State of the Art:

- ▶ **Full analysis of properties of refinement operators** in DLs
- ▶ A **refinement operator** conforming to the theoretical findings
- ▶ **Algorithm handling the unavoidable limitations** of the operator
- ▶ Provision of a **preliminary evaluation**

### Future Work:

- ▶ More **evaluation examples** (performance on noisy or inconsistent data)
- ▶ Create (more) **benchmarks** to assess scalability and enable comparison of different algorithms
- ▶ Tests on real world data, e.g. DBpedia
- ▶ **Embed learning algorithm in ontology editor** e.g. OntoWiki
- ▶ **Extend operator** to other description languages and OWL (nominals, cardinality restrictions, datatype integer)
- ▶ Algorithm performance improvements: using domain/range of properties, subproperties

UNIVERSITÄT LEIPZIG

Computer Science Department  
Universität Leipzig



AIFB Institute  
Universität Karlsruhe