Concept Learning in the $\mathcal{ALC}$ Description Logic

Jens Lehmann[1] and Pascal Hitzler[2]

[1] Department of Computer Science, Universität Leipzig, Germany – lehmann@informatik.uni-leipzig.de
[2] 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 $\top$, $\bot$, 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

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

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

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

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

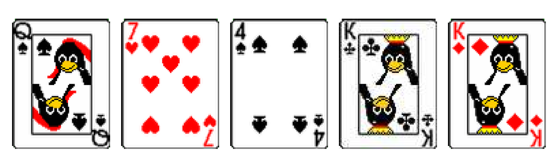
Algorithm

Input: *horizExpFactor* in]0,1]
ST (search tree) is set to the tree consisting only of the root node $(\top, 0, q(\top), \mathit{false})$
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 $(D, n, -, \mathit{checkRed}(ST, D))$ with $D \in \mathit{trans}(\rho^{cl}_\downarrow(C))$ and $|D| = n+1$ as children of N
    evaluate created non-redundant nodes
    change N to $(C, n+1, q, b)$
  end
  $\mathit{minHorizExp} = \max(\mathit{minHorizExp}, \lceil \mathit{horizExpFactor} * (n+1) \rceil)$
  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*

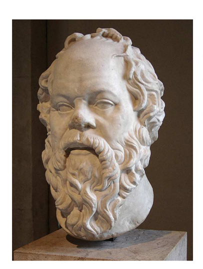
Evaluation

problem	axioms, concepts, roles objects, examples	DL-Learner runtime	DL-Learner length	DL-Learner correct	YinYang runtime	YinYang length	YinYang 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%	-	-	-

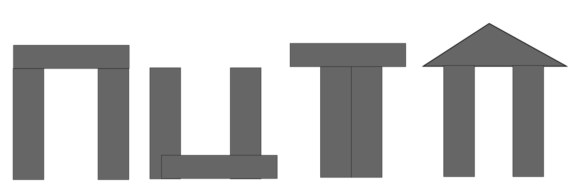
Poker - Pair
∃hasCard.∃sameRank.⊤



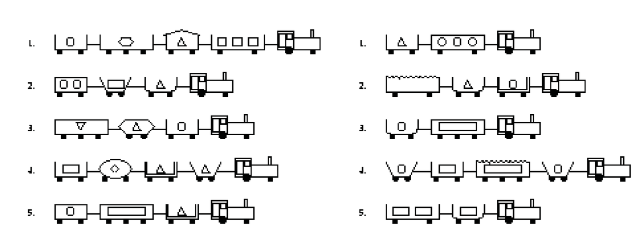
Poker - Straight
∃hasCard.∃nextRank.∃nextRank.∃nextRank.∃nextRank.⊤



Moral Reasoner
simple: Guilty ≡ Blameworthy ⊔ Vicarious_blame
complex: Guilty ≡ ¬Justified ⊓ (Vicarious ⊔ (Negligent ⊓ Responsible))



Arches
Arch ≡ ∃hasPillar.(FreeStandingPillar ⊓ ∃leftOf.∃supports.⊤)



Trains
Left ≡ ∃hasCar.(Closed ⊓ 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