$\mathcal{ALC}$ Concept Learning with Refinement Operators

Jens Lehmann Pascal Hitzler

UNIVERSITÄT LEIPZIG


Universität Karlsruhe (TH)

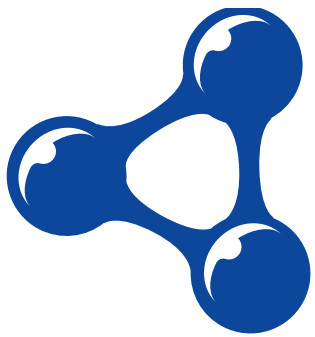
June 17, 2007

Outline

Outline

Introduction to Description Logics

- Description Logics is the name of a family of languages for knowledge representation
- fragment of first order predicate logic
- less expressive power than predicate logic, but decidable inference problems
- intuitive variable free syntax
- basis of the ontology language OWL



Modelling Knowledge in DLs

- representation of knowledge using roles, concepts, and objects

Modelling Knowledge in DLs

- representation of knowledge using roles, concepts, and objects
- objects
  - correspond to constants
  - examples: `MARY`, `JOHN`

Modelling Knowledge in DLs

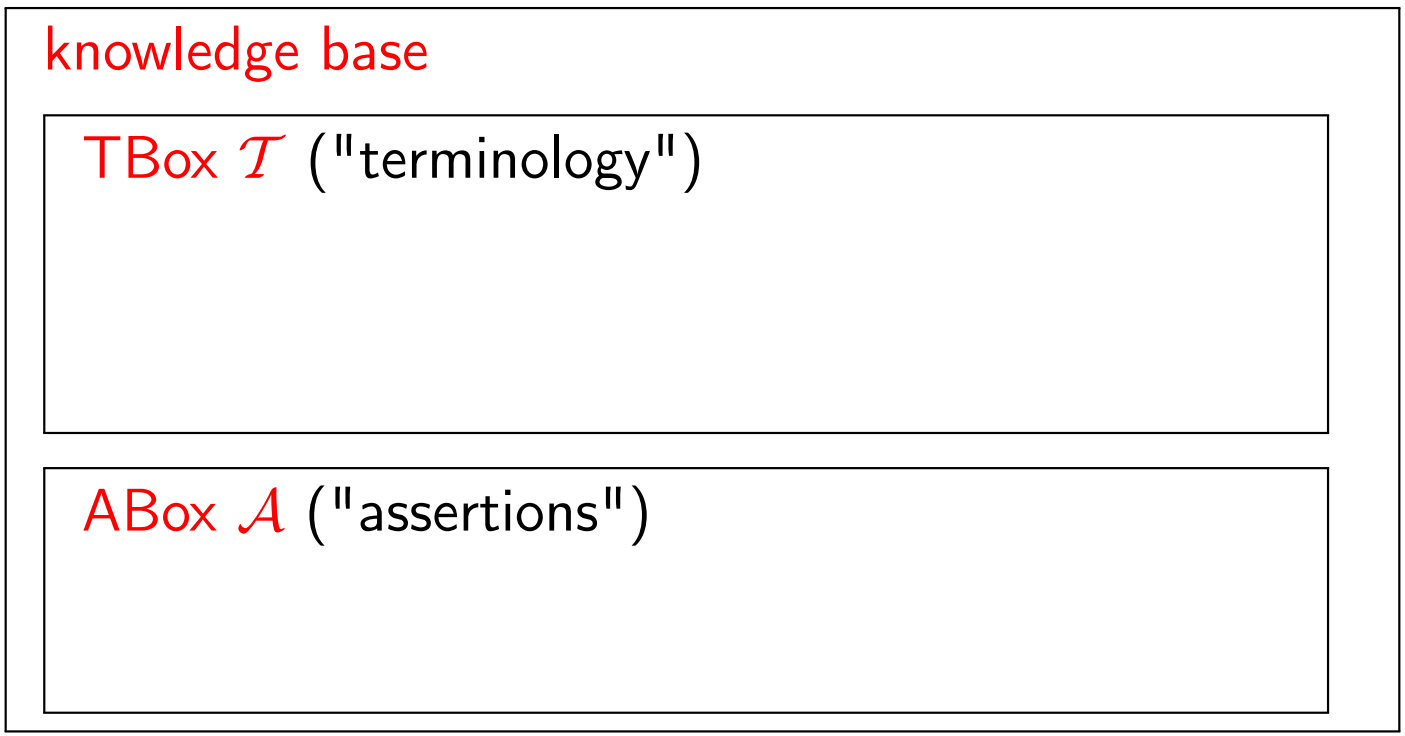
- representation of knowledge using roles, concepts, and objects
- objects
  - correspond to constants
  - examples: `MARY`, `JOHN`
- concepts
  - correspond to unary predicates
  - sets of objects
  - examples: `Student`, `Car`, `Country`

Modelling Knowledge in DLs

- representation of knowledge using roles, concepts, and objects
- objects
  - correspond to constants
  - examples: `MARY`, `JOHN`
- concepts
  - correspond to unary predicates
  - sets of objects
  - examples: `Student`, `Car`, `Country`
- roles
  - corresponds to binary predicates
  - describe connections between objects
  - examples: `hasChild`, `isPartOf`

Example Knowledge Bases

A knowledge base has the following structure:



Example Knowledge Bases

A knowledge base has the following structure:

knowledge base

TBox $\mathcal{T}$ ("terminology"), e.g.

$\texttt{Woman} \equiv \texttt{Human} \sqcap \texttt{Female}$

$\texttt{Mother} \equiv \texttt{Woman} \sqcap \exists \texttt{hasChild}.\top$

$\texttt{HappyFather} \sqsubseteq \texttt{Father} \sqcap \forall \texttt{hasChild}.\texttt{Female}$

ABox $\mathcal{A}$ ("assertions")

Example Knowledge Bases

A knowledge base has the following structure:

knowledge base

TBox $\mathcal{T}$ ("terminology"), e.g.

$\texttt{Woman} \equiv \texttt{Human} \sqcap \texttt{Female}$

$\texttt{Mother} \equiv \texttt{Woman} \sqcap \exists \texttt{hasChild}.\top$

$\texttt{HappyFather} \sqsubseteq \texttt{Father} \sqcap \forall \texttt{hasChild}.\texttt{Female}$

ABox $\mathcal{A}$ ("assertions"), e.g.

$\texttt{Woman}(\texttt{MONICA})$

$\texttt{hasChild}(\texttt{MONICA}, \texttt{JESSICA})$

$\mathcal{ALC}$ Syntax and Semantics

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

Table: $\mathcal{ALC}$ syntax and semantics

Reasoning

- TBox:
  - subsumption between concepts: $C \sqsubseteq_{\mathcal{T}} D$ means, that D is more general than C wrt. $\mathcal{T}$, e.g. $\texttt{Mother} \sqsubseteq_{\mathcal{T}} \texttt{Woman}$
  - equivalence: $C \equiv_{\mathcal{T}} D$ means that two concepts are semantically equivalent
  - strict subsumption: $C \sqsubset_{\mathcal{T}} D$ iff $C \sqsubseteq_{\mathcal{T}} D$ and $C \not\equiv_{\mathcal{T}} D$
  - satisfiability of concepts: $\texttt{Male} \sqcap \texttt{Female}$ unsatisfiable if $\mathcal{T}$ contains $\texttt{Male} \equiv \neg\texttt{Female}$
- ABox (and TBox):
  - consistency: ABox is consistent if there are no contradictions
  - instance check: tests whether an object belongs to a concept
  - retrieval: gets all objects belonging to a concept

OWL

- OWL is an acronym for Web Ontology Language
- 3 flavors: OWL-Lite, OWL-DL, OWL-Full
- OWL-DL is based on the description language $\mathcal{SHOIN}(D)$ (more expressive than $\mathcal{ALC}$)
- W3C recommendation since 2004
- widely used standard for representing knowledge in the Semantic Web (with application areas outside the Web)
- many ontology editors (e.g. Protégé, Swoop, Semantic Works, OntoWiki) and reasoners (e.g. KAON2, Pellet, Racer, FACT++) available for OWL-DL

Outline

Learning Problem

- short version: ILP on DLs
- goal: learn a concept definition from positive examples + negative examples + background knowledge

Learning Problem

- short version: ILP on DLs
- goal: learn a concept definition from positive examples + negative examples + background knowledge
- we have a target concept name `Target` and a knowledge base $\mathcal{K}$ as background knowledge
- examples are of the form `Target`(a), where a is an object
- let E^+ be the set of positive examples and E^- the set of negative examples
- we want to find a definition Def of the form `Target` $\equiv C$ such that for $\mathcal{K}' = \mathcal{K} \cup \{Def\}$ we have $\mathcal{K}' \models E^+$ and $\mathcal{K}' \not\models E^-$

Application Areas

Why is it useful to learn in DLs?

- may have similar applications like ILP (Inductive Logic Programming) approaches for learning horn clauses e.g. in biology and medicine where ontologies are widely used
- incremental ontology learning in context of OWL and the Semantic Web – make it easier for users to build ontologies from existing data

Outline

Refinement Operators - Definitions

- consider quasi-ordered space $(S, \preceq)$, i.e. $\preceq$ is reflexive and transitive
- downward (upward) refinement operator ρ is a mapping from S to 2^S such that for any $C \in S$:

$$C' \in \rho(C) \text{ implies } C' \preceq C \quad (C \preceq C')$$

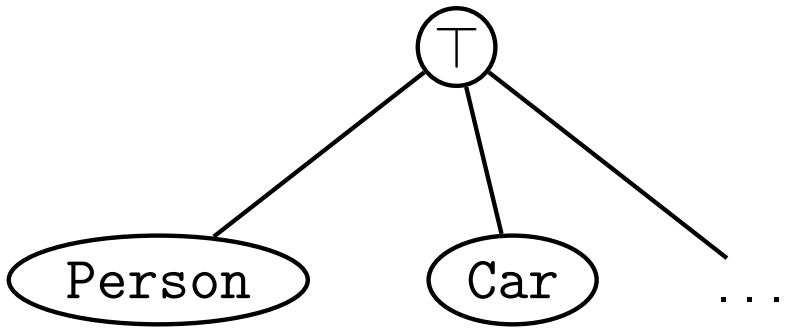
- refinement operator in the quasi-ordered space $(\mathcal{L}, \sqsubseteq_{\mathcal{T}})$ is called an $\mathcal{L}$ refinement operator
- instead of $D \in \rho(C)$ we often write $C \rightsquigarrow_\rho D$, e.g. $\top \rightsquigarrow_\rho \texttt{Male} \rightsquigarrow_\rho \texttt{Male} \sqcap \exists \texttt{hasChild}.\top$

Learning with Refinement Operators

- refinement operator can be used to span up a search tree
- refinement operator + search heuristic = learning algorithm

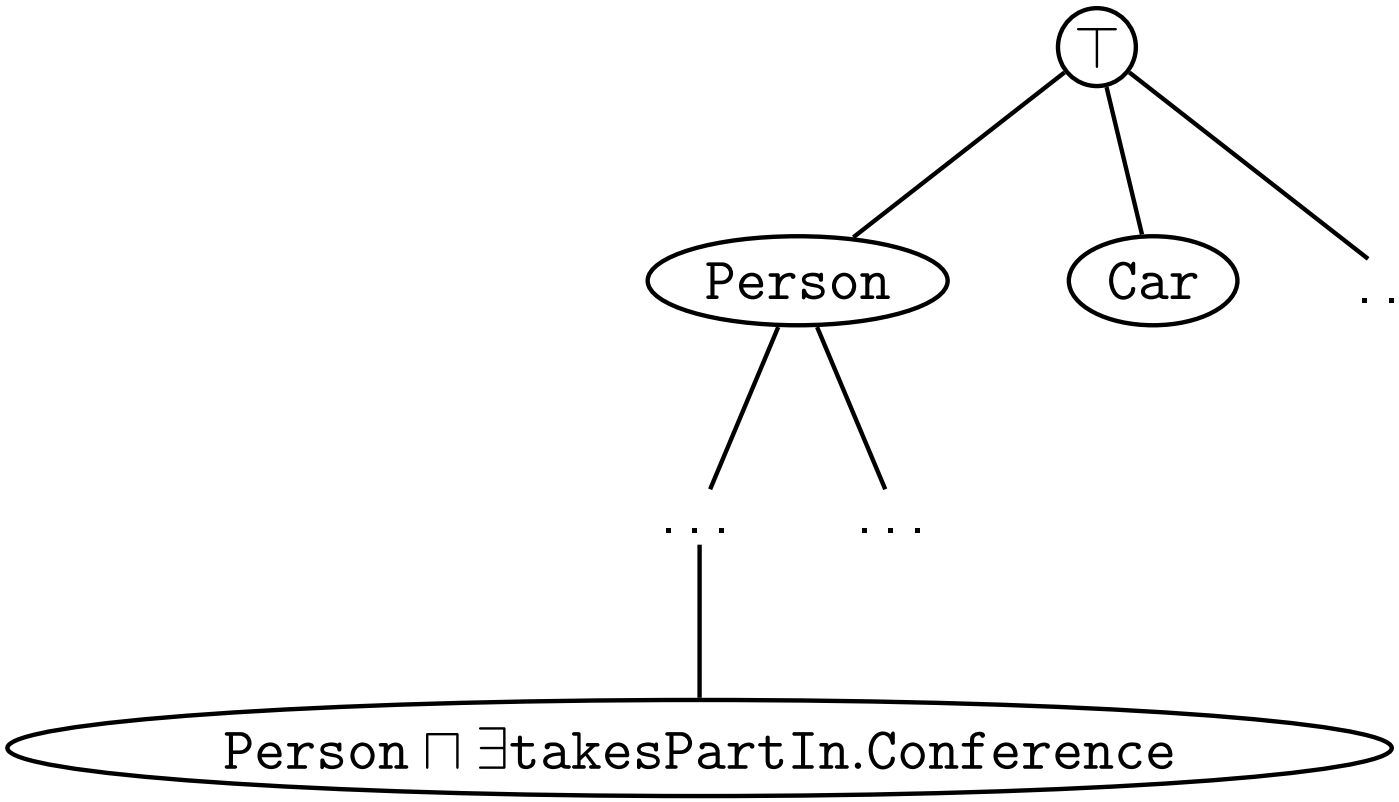
Learning with Refinement Operators

- refinement operator can be used to span up a search tree
- refinement operator + search heuristic = learning algorithm



Learning with Refinement Operators

- refinement operator can be used to span up a search tree
- refinement operator + search heuristic = learning algorithm



Properties of Refinement Operators

An $\mathcal{L}$ refinement operator ρ is called

- finite iff $\rho(C)$ is finite for any concept C.
- redundant iff there exist two different refinement chains from a concept C to a concept D.
- proper iff for any concepts C and D, $D \in \rho(C)$ implies $C \not\equiv_{\mathcal{T}} D$.

An $\mathcal{L}$ downward refinement operator is called

- complete iff for any concepts C and D with $C \sqsubset_{\mathcal{T}} D$ we can reach a concept E with $E \equiv_{\mathcal{T}} C$ from D by ρ.
- weakly complete iff for any concept C with $C \sqsubset_{\mathcal{T}} \top$ we can reach a concept E with $E \equiv_{\mathcal{T}} C$ from $\top$ by ρ.
- minimal iff for all C, $\rho(C)$ contains only downward covers and all its elements are incomparable with respect to $\sqsubseteq$

Research Task

- we researched the properties (completeness, properness, redundancy, finiteness, minimality) of refinement operators
- key question: Which properties can be combined?
- 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)

Minimality

- Do covers exist in expressive DLs?

Minimality

- Do covers exist in expressive DLs?
- Yes. The following is a downward cover of the $\top$ concept:

$$\bigsqcup_{r \in N_R} \exists r.\top \sqcup \bigsqcup_{A \in N_C} A$$

Minimality

- Do covers exist in expressive DLs?
- Yes. The following is a downward cover of the $\top$ concept:

$$\bigsqcup_{r \in N_R} \exists r.\top \sqcup \bigsqcup_{A \in N_C} A$$

- however, minimality is unlikely to play a central role:
  - finding and constructing covers is hard and they probably do not provide a sufficient generalisation leap
  - even in less expressive DL languages like $\mathcal{AL}$ minimal operators cannot be weakly complete

Proposition

There exists no minimal and weakly complete $\mathcal{AL}$ downward refinement operator.

No Ideal Operators

Proposition

For any considered language $\mathcal{L}$, there does not exist any ideal $\mathcal{L}$ refinement operator.

Example

- assume finite, proper downward refinement operator ρ with $\rho(\top) = \{C_1, \ldots, C_n\}$ exists
- let m be greater than the quantor depth of any concept in $\rho(\top)$
- the following concept cannot be reached from $\top$:

$$D = \underbrace{\forall r. \ldots . \forall r}_{m-\text{times}} . \bot \sqcup \underbrace{\exists r. \ldots . \exists r}_{(m+1)-\text{times}} . \top$$

Positive Results

Proposition

For any considered language $\mathcal{L}$, there exists a complete and finite $\mathcal{L}$ refinement operator.

- we have built a system integrating a complete and finite $\mathcal{ALC}$ refinement operator in a Genetic Programming framework

Positive Results

Proposition

For any considered language $\mathcal{L}$, there exists a complete and finite $\mathcal{L}$ refinement operator.

- we have built a system integrating a complete and finite $\mathcal{ALC}$ refinement operator in a Genetic Programming framework

Proposition

For any considered language $\mathcal{L}$, there exists a complete and proper $\mathcal{L}$ refinement operator.

- will be used in the algorithm presented later on

Redundancy

Proposition

For any considered language $\mathcal{L}$, there exists a complete and non-redundant $\mathcal{L}$ refinement operator.

Redundancy

Proposition

For any considered language $\mathcal{L}$, there exists a complete and non-redundant $\mathcal{L}$ refinement operator.

- but: this result is achieved by using the countable infiniteness of the set of all concepts
- a negative result can be shown under the following mild assumption (for downward refinement): $\bot \in \rho^*(C)$ for any concept C

Proposition

Let $\mathcal{L}$ be a considered language and ρ a refinement operator satisfying the assumption above. Then ρ is not complete and non-redundant.

Redundancy

Proposition

For any considered language $\mathcal{L}$, there exists a complete and non-redundant $\mathcal{L}$ refinement operator.

- but: this result is achieved by using the countable infiniteness of the set of all concepts
- a negative result can be shown under the following mild assumption (for downward refinement): $\rho^*(C)$ contains only finitely many different concepts equivalent to $\bot$

Proposition

Let $\mathcal{L}$ be a considered language and ρ a refinement operator satisfying the assumption above. Then ρ is not complete and non-redundant.

Redundancy

Proposition

For any considered language $\mathcal{L}$, there exists a complete and non-redundant $\mathcal{L}$ refinement operator.

- but: this result is achieved by using the countable infiniteness of the set of all concepts
- a negative result can be shown under the following mild assumption (for downward refinement): concepts C_{up}, C_{down} with $C_{down} \sqsubset C_{up}$, $\{C \mid C \in \rho^*(C_{up}), C \equiv C_{down}\}$ finite, and an infinite set S of pairwise incomparable concepts strictly subsumed by C_{up} and strictly subsuming C_{down} exists

Proposition

Let $\mathcal{L}$ be a considered language and ρ a refinement operator satisfying the assumption above. Then ρ is not complete and non-redundant.

Theorem about Properties of $\mathcal{L}$ Refinement Operators

Theorem (properties of $\mathcal{L}$ refinement operators)

Considering the analysed properties and languages $\mathcal{L}$, the following are maximal sets of properties 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*}

Outline

Step 1: Define an Operator

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

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

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

Completeness of $\rho_\downarrow$

Proposition (completeness of $\rho_\downarrow$)

$\rho_\downarrow$ is complete.

Proof Idea:

- *first show weak completeness:*
  - *a set $S_\downarrow$ of $\mathcal{ALC}$ concepts was defined (see article for the definition of $S_\downarrow$)*
  - *for every $\mathcal{ALC}$ concept there exists an equivalent concept in $S_\downarrow$*
  - *all concepts in $S_\downarrow$ can be reached by $\rho_\downarrow$ from $\top$*
- *prove completeness using the weak completeness result*

Infiniteness of $\rho_\downarrow$

- $\rho_\downarrow$ is infinite, e.g. there are infinitely many refinement steps of the form:

$$\top \rightsquigarrow_{\rho_\downarrow} \underbrace{\forall \texttt{hasChild}. \ldots \forall \texttt{hasChild}}_{\text{arbitrarily often}} . \texttt{Male}$$

- solution: we only consider refinements up to length n of concepts (there are only finitely many of these)
- n is initially set to 0 and increased by the learning algorithm as needed

Properness

- $\rho_\downarrow$ is not proper: $\top \rightsquigarrow_{\rho_\downarrow} \exists \texttt{hasChild}.\top \sqcup \forall \texttt{hasChild}.\texttt{Male}$
- idea: consider the closure $\rho_\downarrow^{cl}$ of $\rho_\downarrow$:
  $D \in \rho_\downarrow^{cl}(C)$ iff there exists a refinement chain

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

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

Proposition

For any concept C in negation normal form and any natural number n the set

$$\{D \mid D \in \rho_\downarrow^{cl}(C), |D| \leq n\}$$

can be computed in finite time.

Redundancy

$\rho^{cl}_{\downarrow}$ is redundant:

$$\begin{array}{ccc} \forall r_1.A_1 \sqcup \forall r_2.A_1 & \rightsquigarrow_{\rho_\downarrow} & \forall r_1.(A_1 \sqcap A_2) \sqcup \forall r_2.A_1 \\ \uparrow_d \swarrow & & \uparrow_d \swarrow \\ \forall r_1.A_1 \sqcup \forall r_2.(A_1 \sqcap A_2) & \rightsquigarrow_{\rho_\downarrow} & \forall r_1.(A_1 \sqcap A_2) \sqcup \forall r_2.(A_1 \sqcap A_2) \end{array}$$

- redundancies should be detected by the learning algorithm
- result in paper: we can check whether an occurring concept is redundant with respect to a search tree in polynomial time

Step 2: DL-Learner Algorithm

Input: *horizExpFactor* in]0,1]

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

2 $\mathit{minHorizExp} = 0$

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

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

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

6 **begin**

7 add all nodes $(D, n, -, \mathit{checkRed}(ST, D))$ with $D \in \mathit{trans}(\rho^{cl}_{\downarrow}(C))$ and $|D| = n + 1$ as children of N

8 evaluate created non-redundant nodes

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

10 **end**

11 $\mathit{minHorizExp} = \max(\mathit{minHorizExp}, \lceil \mathit{horizExpFactor} * (n + 1)) \rceil)$

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

13 expand these nodes up to *minHorizExp*

14 Return a correct concept in ST

Simple Example

$\texttt{Male} \equiv \neg\texttt{Female}$	$\texttt{Male}(\texttt{MARC})$
	$\texttt{Male}(\texttt{STEPHEN})$
$\texttt{hasChild}(\texttt{STEPHEN},\texttt{MARC})$	$\texttt{Male}(\texttt{JASON})$
$\texttt{hasChild}(\texttt{MARC},\texttt{ANNA})$	$\texttt{Male}(\texttt{JOHN})$
$\texttt{hasChild}(\texttt{JOHN},\texttt{MARIA})$	$\texttt{Female}(\texttt{ANNA})$
$\texttt{hasChild}(\texttt{ANNA},\texttt{JASON})$	$\texttt{Female}(\texttt{MARIA})$
	$\texttt{Female}(\texttt{MICHELLE})$

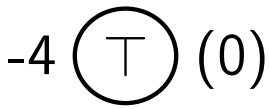
positive:$\{\texttt{STEPHEN}, \texttt{MARC}, \texttt{JOHN}\}$

negative:$\{\texttt{JASON}, \texttt{ANNA}, \texttt{MARIA}, \texttt{MICHELLE}\}$

possible solution: $\texttt{Male} \sqcap \exists \texttt{hasChild}.\top$

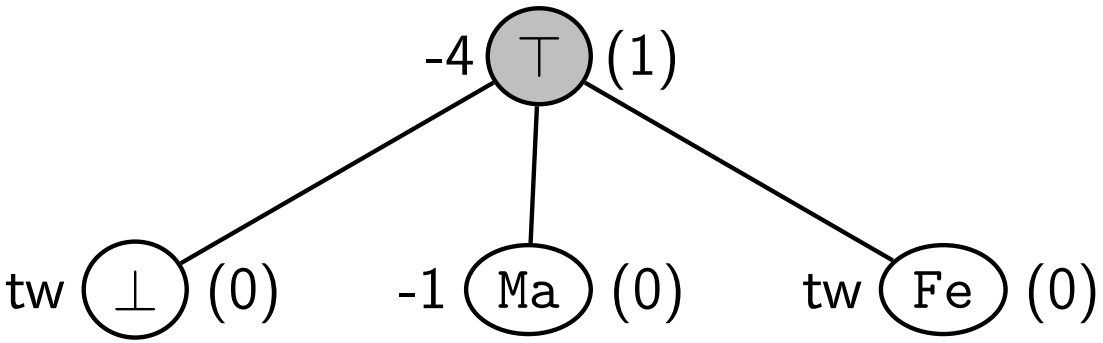
Simple Example

Initialisation (minimum horizontal expansion = 0):



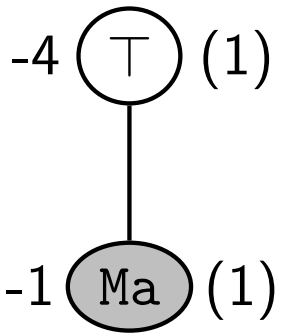
Simple Example

Step 1 (minimum horizontal expansion = 1):



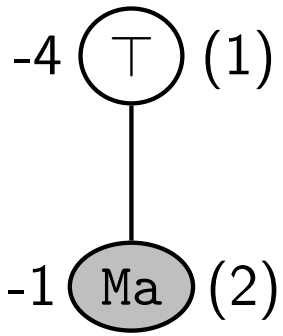
Simple Example

Step 2 (minimum horizontal expansion = 1):



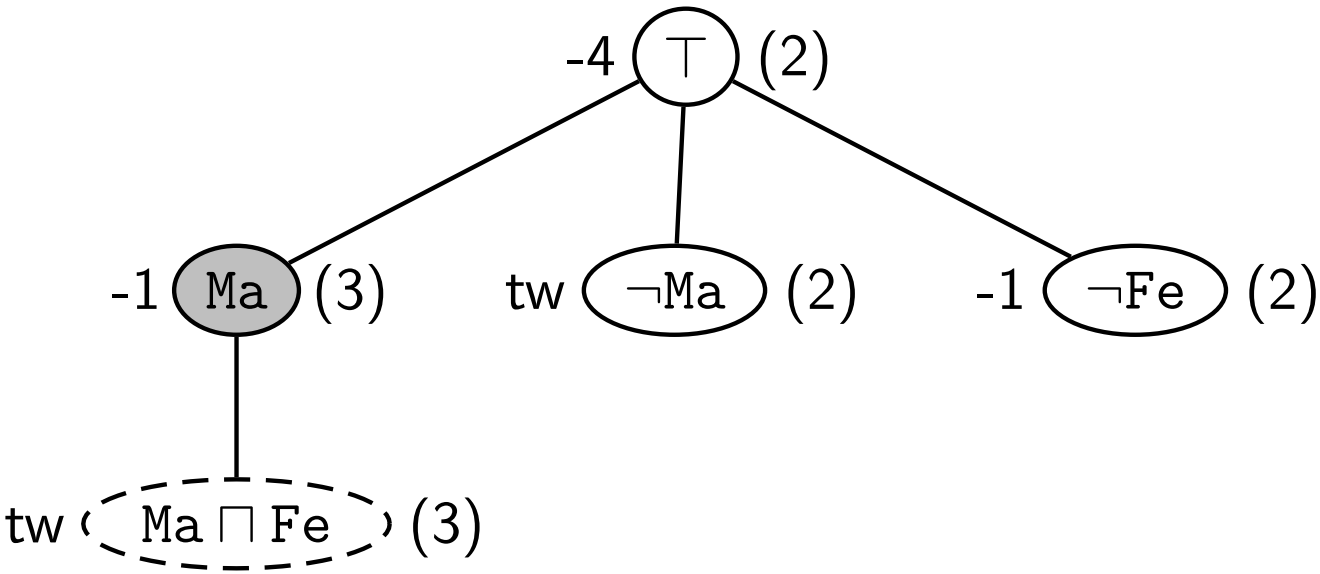
Simple Example

Step 3 (minimum horizontal expansion = 1):



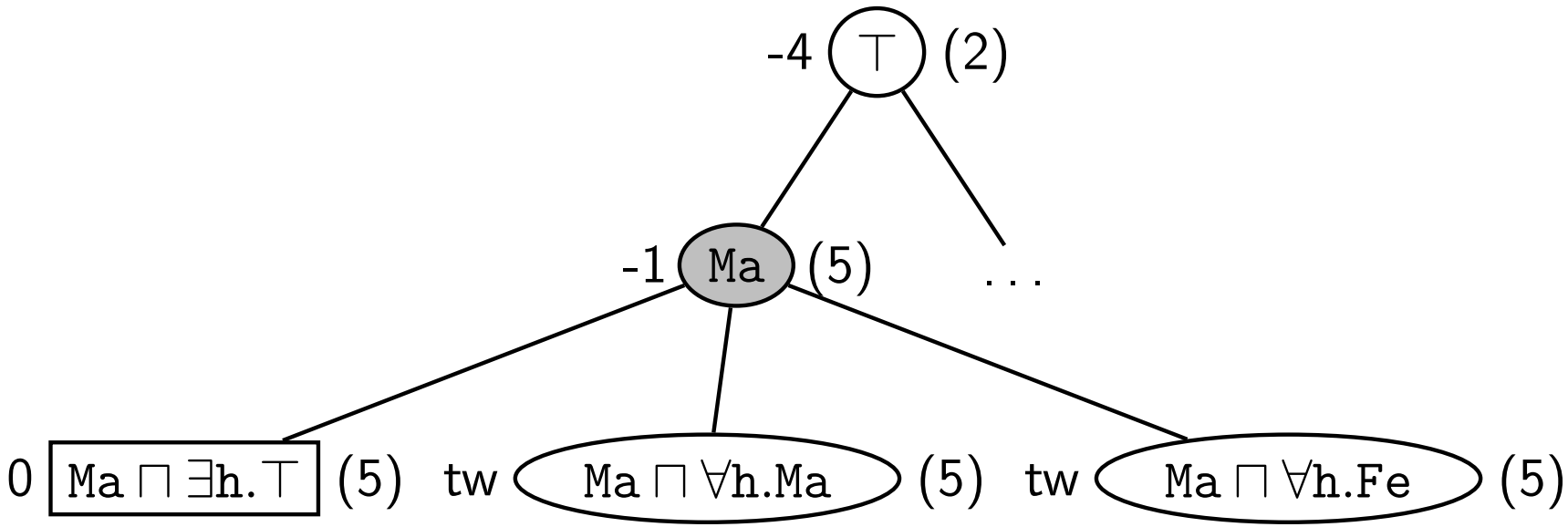
Simple Example

Step 4 (minimum horizontal expansion = 2):



Simple Example

Step x (minimum horizontal expansion = 2):



solution: $\texttt{Male} \sqcap \exists \texttt{hasChild}.\top$

Outline

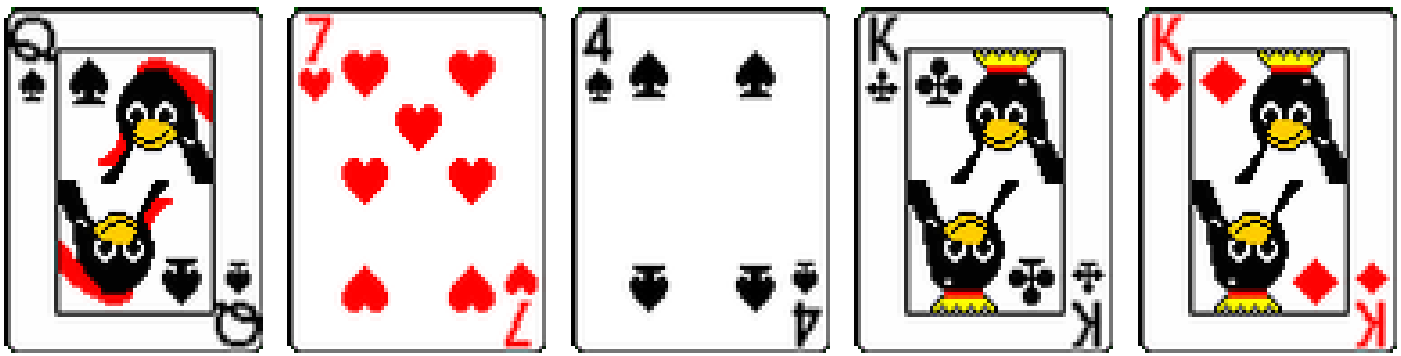
Evaluation

- not much evaluation examples or benchmarks available for concept learning from examples yet
- examples had to be converted from existing ones e.g. in the UCI Machine Learning Repositories
- evaluation system: 1.4GHz, DIG 1.1 Interface, Pellet 1.4RC1 reasoner, horiz. expansion factor 0.6

Poker Examples

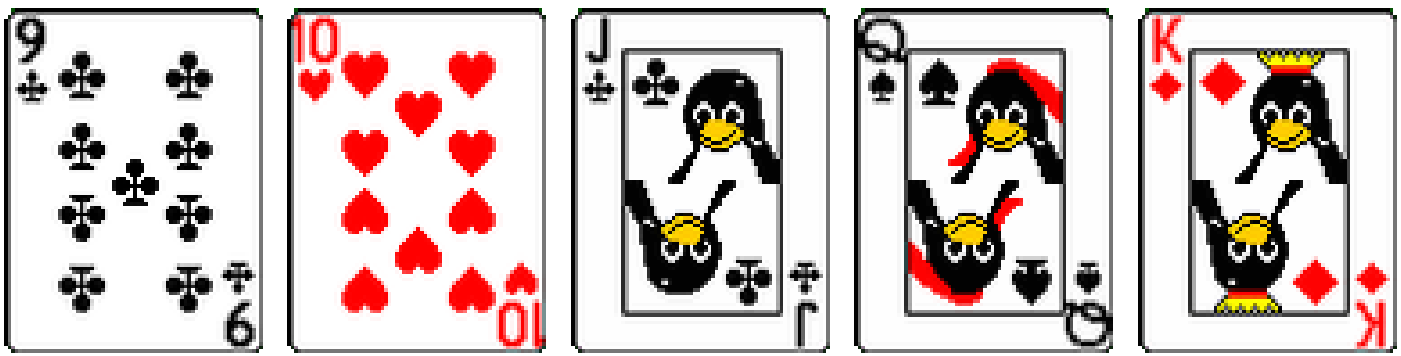
Poker - Pair

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

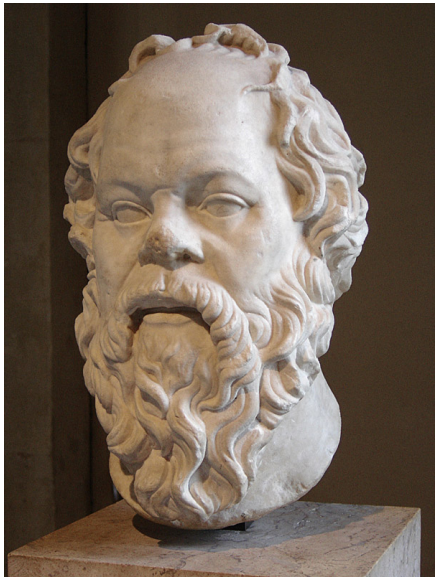


Poker - Straight

$\exists \texttt{hasCard}.\exists \texttt{nextRank}.\exists \texttt{nextRank}.\exists \texttt{nextRank}.\exists \texttt{nextRank}.\top$



Moral Reasoner and Arch Examples



Moral Reasoner

simple variant:

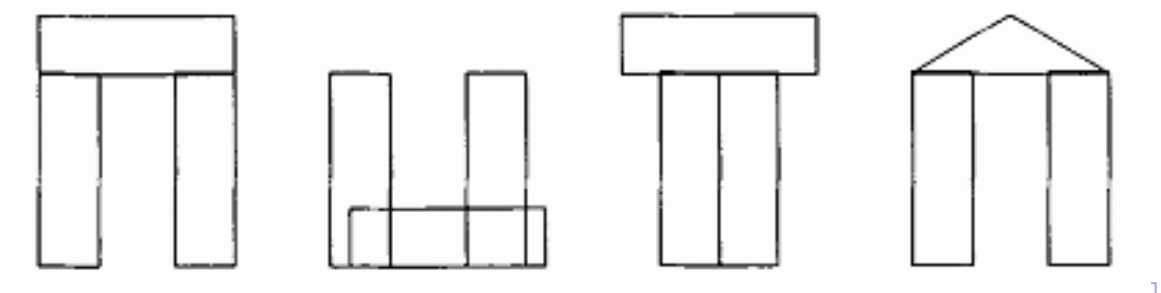
$$\texttt{Guilty} \equiv \texttt{Blameworthy} \sqcup \texttt{Vicarious\_blame}$$

complex variant:

$$\texttt{Guilty} \equiv \neg\texttt{Justified} \sqcap (\texttt{Vicarious} \sqcup (\texttt{Negligent\_c} \sqcap \texttt{Responsible}))$$

Arches

$$\texttt{Arch} \equiv \exists \texttt{hasPillar}.(\texttt{FreeStandingPillar} \sqcap \exists \texttt{leftOf}.\exists \texttt{supports}.\top)$$



Evaluation Table

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

- DL-Learner finds solutions for the given problems
- examples cover different complexity and size of background knowledge
- . . . and different concept constructors in solutions
- most of the time spend for reasoner requests

Evaluation Table

problem	DL-Learner			YinYang		
	runtime	length	correct	runtime	length	correct
trains	1.1s	5	100%	2.3s	8	100%
arches	4.6s	9	100%	1.5s	23	100%
moral (simple)	17.7s	3	100%	205.3s	69	67.4%
moral (complex)	88.1s	8	100%	181.4s	70	69.8%
poker (pair)	7.7s	5	100%	17.1s	43	100%
poker (straight)	35.6s	11	100%	-	-	-

- YinYang only available system to the best of our knowledge
- DL-Learner tries to find shorter solutions (likely to be better according to Occam's Razor)

Evaluation Table

problem	DL-Learner			YinYang		
	runtime	length	correct	runtime	length	correct
trains	1.1s	5	100%	2.3s	8	100%
arches	4.6s	9	100%	1.5s	23	100%
moral (simple)	17.7s	3	100%	205.3s	69	67.4%
moral (complex)	88.1s	8	100%	181.4s	70	69.8%
poker (pair)	7.7s	5	100%	17.1s	43	100%
poker (straight)	35.6s	11	100%	-	-	-

- YinYang only available system to the best of our knowledge
- DL-Learner tries to find shorter solutions (likely to be better according to Occam's Razor)

Outline

Contributions to the State of the Art

- full analysis of properties of refinement operators in DLs
- a refinement operator conforming to the theoretical findings
- an algorithm handling the unavoidable limitations of the operator
- provision of examples and a preliminary evaluation

Future Work

- more evaluation examples, e.g. asses performance on noisy or inconsistent data
- create (more) benchmarks to assess scalability and enable easier comparison between different algorithms
- tests on real world data, e.g. DBpedia
- embed learning algorithm in ontology editor e.g. OntoWiki
- extend algorithm to other description languages and OWL (cardinality restrictions, datatype integer)
- algorithm performance improvements: using domain/range restrictions, subproperty relationships

Thank you for your attention.

contact:
lehmann@informatik.uni-leipzig.de