Knowledge-Based Systems (KBS)

Machine Learning and Knowledge Engineering

Winter semester 2021/2022

Prof. Dr. Martin Atzmüller

Osnabrück University & DFKI
Semantic Information Systems Group
https://sis.cs.uos.de/

2 – KBS – Knowledge Engineering and Representation Formalisms

1. Overview: Inference and Knowledge
2. Fundamentals of Knowledge Engineering
3. Knowledge Representation – Basic Principles
4. Basic Techniques of Knowledge Representation
5. Knowledge Representation By (Three) Example(s)

Before we start: About Inference

- Given a knowledge base, how can we infer new knowledge?
- Can we generate/learn a knowledge base from observations?
- What about explanations? Can this somehow be integrated?

Coming back to making conclusions from existing knowledge.

- How do humans do this?
- And how can we model it in a KBS?

KBS – About Inference

- Inference relation: R
- Knowledge: K
- New (inferred) knowledge: O
- With: $(K, O) \in R$

Examples:

- Given some knowledge (formalized in a knowledge base), we can then infer new knowledge
- Given some observations, we can hypothesize some "more general" knowledge
- Given some observations, we can generate explanations (potentially also utilizing some formalized knowledge)

Inference: Basic Computational Modeling

Core element of a KBS: Knowledge base

- Inference: Relation between formalized (given) knowledge, and new (inferred) knowledge – respectively, their knowledge representations
- Syntax of KR language: How to build sentences in the language
- Semantics: "Meaning", i. e., which concepts of the world to represent do the sentences relate to
- Example: $x \leq y$

Modeling:

- With K, O given by syntactic elements, then R is a binary relation on the syntactic level
- $(K, O) \in R$ iff. a human infers *semantics(O)* given *semantics(K)*

Perspectives on R

- *Deduction*: Given K we can infer O given R
- *Abduction*: Given O provide an explanation (K) for O
- *Test*: Check if O logically follows from K
- *Induction*: Learn from a set of observations O some new K, i. e., in a (rule-based) representation

- Original proposal of specific inference types according to [Peirce 1931]: Deduction, Abduction, Induction
- However, many more possible, e. g., relating to inference from uncertain knowledge, etc.

Knowledge Engineering

What is Knowledge Engineering?

- "Knowledge engineering (KE) refers to all technical, scientific and social aspects involved in building, maintaining and using knowledge-based systems." (Wikipedia)
- "Knowledge engineering is a field within artificial intelligence that develops knowledge-based systems. Such systems are computer programs that contain large amounts of knowledge, rules and reasoning mechanisms to provide solutions to real-world problems. [...]" (https://www.igi-global.com/dictionary/)

From Data to Information and Knowledge

So, what about knowledge?
Our working definition of knowledge:

- organized/structured information, with a
- declarative representation

Some questions to start with:

- What is data?
- What is then information?
- What, ultimately is knowledge?
- Is there a relationship between those?
- If so, how can we characterize this relationship?

Data, Information and Knowledge

From [Davenport & Prusak, 1998]:

- “*Data* is a set of discrete, objective facts about events. [....] Data by itself has little relevance or purpose.”
- “*Information* is a message, usually in the form of a document or an audible or visible communication. Information is meant to change the way the receiver perceives something, to have an impact on his judgment and behavior. It must inform; it is data that makes a difference.”

Knowledge

From [Davenport & Prusak, 1998]:

- "*Knowledge* is a fluid mix of framed experience, values, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information. In organizations, it often becomes embedded not only in documents or repositories but also in organizational routines, processes, practices, and norms."
- So, according to [Devlin 1999]:
  - Data = symbol + syntax
  - Information = data + meaning
  - Knowledge = internalized information + ability to use it.

KR Principles

Five Principles of Knowledge Representation Formalisms

- "A knowledge representation is
  - a surrogate,
  - a medium of human expression,
  - a set of ontological commitments,
  - a fragmentary theory of intelligent reasoning,
  - and a medium for pragmatically efficient computation."

  [Davis, Shrobe, Szolovits, 1993]
- They way, in which each type of knowledge representation addresses these principles, characterizes their "spirit". Each knowledge representation somehow addresses these (partially contradicting positions) in a specific way.

KR as a Surrogate

- “Knowledge representation is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting. [...]
- Reasoning is a process that goes on internally [of a person or program], while most things it wishes to reason about exist only externally.”
  [Davis, Shrobe, Szolovits, 1993]

KR as a Medium of Human Expression

- "Knowledge representations are [...] the medium of expression and communication in which we tell the machine (and perhaps one another) about the world. [...] Knowledge representation is thus a medium of expression and communication for the use by us. [...] A representation is the language in which we communicate, hence we must be able to speak it without heroic effort." [Davis, Shrobe, Szolovits, 1993]
- [Davis, Shrobe, Szolovits, 1993] ask: "What things are easily said in the language and what kind of things are so difficult as to be pragmatically impossible?"

KR as Ontological Commitment

- A knowledge representation
  - “is a set of ontological commitments, i.e., an answer to the following question: ‚In what terms should we think about the world? [...]
  - In selecting any representation, we are [...] making a set of decisions about how and what to see in the world. [...]
  - We (and our reasoning machines) need guidance in deciding what in the world to attend to, and what to ignore.“

  [Davis, Shrobe, Szolovits, 1993]

KR as Fragmentary Theory of Intelligent Reasoning

- "The initial conception of a knowledge representation is typically motivated by some insight indicating how people reason intelligently, or by some belief about what it means to reason intelligently at all."
  [Davis, Shrobe, Szolovits, 1993]

- The authors mention five areas/perspectives, which discuss intelligent problem solving:
  - Mathematical Logics
  - Psychology
  - Biology
  - Statistics
  - Economics

Models of Knowledge

These different perspectives on knowledge representation lead to different models of knowledge:

- Biology: Networks – Neural Networks
- Mathematical Logics: Deduction – Logical Calculus, Prolog
- Statistics – Uncertainty: Fuzzy-Logics, Bayesian Networks
- Philosophy/Psychology: Semantic Networks, Frames, Ontologies, Knowledge Graphs
- Economics: Goals – Case-Based Reasoning, Agents

Models of Knowledge

- Symbolic Representations
  - Symbolic representation are surrogats for things of the (external) world.
  - Manipulation via inference processes
  - Advantages:
    - Knowledge is captured via a formal representation
    - Representations are readable and meaningful
- Non-symbolic representations
  - Examples: Analog maps and diagrams, neural networks
  - Advantages:
    - Often fewer assumptions need to be made
    - Non-symbolic representations can often deal better with imprecise knowledge

KR as a Medium for Efficient Computation

Knowledge representation

- „is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished.
- One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.“

[Davis, Shrobe, Szolovits, 1993]

Expressiveness vs. Efficiency

Fundamental Trade-off: Expressiveness vs. Efficiency (!)

Desired Properties

- Expressive representation, complete inference procedures
- Efficient computation (tractability: polynomial complexity)

However: Both does not go together!

Example: PL1 is expressive, but does not provide efficient computation procedures. This trade-off exists for knowledge representation in general (Levesque & Brachman, 1985).

Expressiveness vs. Efficiency – Approaches

- Expressive/general representations using approximate inference methods
- Specialized representations (targeting a specific domain)
- Potentially: multiple representations targeting a domain (individual specializations)

- Examples:
  - Datalog (e. g., vs. Prolog)
  - Answer Set Programming
  - OWL – OWL Lite, OWL DL, OWL Full
    (OWL = Web Ontology Language)

KR Semantics

Semantics of Knowledge Representation Formalisms/Languages

- KR languages enable the formal modeling of knowledge
- Its *semantics*, i. e., the “meaning” of the individual language constructs, can be defined in different ways:
    - Operational semantics: The semantics are defined via algorithms, working on language constructs (early semantic networks/frames)
    - Semantic equivalence: Translation in KR formalisms with known semantics (e. g., Frames $\rightarrow$ PL1)

Declarative Semantics

- Syntactic KR structures are related to elements of abstract structures via an "interpretation" (function)
- Example: Set theoretic semantics for PL1
- Advantages:
  - Consistency of a knowledge base can be formally captured (and tested)
  - Subsumption relations can be computed (based on extensions: Extension of one concept is subset of extension of another concept)
  - Correctness and completeness of inference methods can be defined (e. g., for calculating subsumption relations)
- Disadvantage: Semantics is only defined extensionally, intensional aspects are not captured ("morning star", "evening star" and Venus have same meaning/semantics; also – "round rectangle" and "unicorn")

Knowledge Representation Basics

Basic techniques of knowledge representation

- Logic (First-Order/Predicate Logic, PL1)
- Rules
- Objects/Frames
- Constraints
- Probabilistic Reasoning

First-Order Logic

First-Order Logic (PL1) – also called *Predicate Logic*, *First-Order Predicate Calculus*

- Advantages for KR:
    - Well-known, formal notation
    - Domain can be described via axioms
    - Formal semantics!
    - Inference via deduction
- Disadvantages:
    - PL1 is not decidable (It is undecidable, whether a formula of PL1 is valid)
    - High complexity of inference processes
    - Not expressive enough for some applications

Brief Recap: First-Order/Predicate Logic (PL1)

- Example statements:

$$x > 2, \quad x = y + 7, \quad x + y = z$$

- Important: truth value – no meaning without values of x, y, z.
- But: We *can* make propositions given such statements.
- A *predicate* is a property that is affirmed or denied about a *subject* (also called: variable, argument) of a *statement*.
- Example:

$$\text{“}\underbrace{x}_{\text{subject}}\ \underbrace{\text{is greater than 3”}}_{\text{predicate}}$$

- Functional symbol for predicate (P); subject (x) as an argument (to the symbol): $P(x)$

Propositional Function (Predicate)

Definition 1

We call a statement of the form $P(x_1, x_2, \ldots, x_n)$ a *propositional function* P. Here, $(x_1, x_2, \ldots, x_n)$ is an n-tuple and P is a predicate.

A propositional function is a function that

- evaluates to true or false;
- takes one or more arguments;
- expresses a *predicate* involving the argument(s);
- becomes a proposition, whenever values are assigned to the arguments (also, cf. when those are "bound").

Universe of discourse: set of all things to express (about); i. e., set of all (valid) objects that can be assigned to a variable in a propositional function.

Also: *Function symbols* which are used to build more complex expressions – but are no *predicates* (!)

Quantifiers

- Predicate $\rightsquigarrow$ proposition: assign fixed values.
- Another way: *quantification*. Then, predicate true (or false) for
  - *all* possible values in the universe of discourse, or for
  - *some* value(s) in the universe of discourse.
- Respective *Quantification* – two *quantifiers*:
  - *Universal* quantifier ($\forall$), e. g.,

$$\forall x(Q(x) \rightarrow P(x))$$

  - *Existential* quantifier ($\exists$), e. g.,

$$\exists x \exists y P(x)$$

Universal Quantifier

Definition 2

We define the *universal quantification* of a predicate $P(x)$ as: "$P(x)$ is true for all values of x in the universe of discourse."

In notation:

$$\forall x P(x)$$

which can be read "for all x"

If the universe of discourse is finite, e. g., $\{n_1, n_2, \ldots, n_k\}$, then the universal quantifier is simply the conjunction of all of its elements:

$$\forall x P(x) \iff P(n_1) \land P(n_2) \land \cdots \land P(n_k)$$

Existential Quantifier

Definition 3

We define the *existential quantification* of a predicate $P(x)$ as: "There exists an x in the universe of discourse such that $P(x)$ is true."

In notation:

$$\exists x P(x)$$

which can be read "there exists an x"

If the universe of discourse is finite, e. g., $\{n_1, n_2, \ldots, n_k\}$, then the existential quantifier is simply the disjunction of all of its elements:

$$\exists x P(x) \iff P(n_1) \vee P(n_2) \vee \cdots \vee P(n_k)$$

Mixing/Reordering Quantifiers

- Existential and universal quantifiers can be used together, e. g.,

$$\forall x \exists y P(x, y)$$

- However: Read left to right (!)
- $\forall x \exists y P(x, y)$, for example, not equivalent to $\exists y \forall x P(x, y)$
- N.B.: Ordering is important:
  - $\forall x \exists y Loves(x, y)$: everybody loves somebody
  - $\exists y \forall x Loves(x, y)$: There is someone loved by everyone

Binding Variables

- When a quantifier is used on a variable x, x is called *bound*
- If no quantifier is used on a variable in a predicate statement, then x is called *free*
- Example:
  - $\exists x \forall y P(x, y)$: both x and y are bound
  - $\forall x P(x, y)$: x is bound, y is free
- In a *well-formed formula*, all variables are properly quantified
- We call the set of all variables bound by a common quantifier its *scope*

Negation

Negation can also be used with quantified expressions.

Lemma 4

Let $P(x)$ be a predicate. Then the following hold.

$$\neg \forall x P(x) \equiv \exists x \neg P(x)$$

$$\neg \exists x P(x) \equiv \forall x \neg P(x)$$

$\rightsquigarrow$ quantified version of De Morgan's Law

PL1 - Syntax Overview

Symbols:	•	constants	A, B, C
	•	variables	x, y, z
	•	predicate symbols	p, q
	•	function symbols	f, g
	•	relation symbols	$\wedge, \vee, \neg, \rightarrow$
	•	quantifiers	$\forall, \exists$
Terms:	•	constants	A, B
	•	variables	x, y
	•	function symbols, applied to correct number of terms	e.g., $(fx(fBB))$
Formulas:	•	predicate symbols, applied to correct number of terms (atomic formulas)	e.g., $(pxy(fxB))$
	•	If r and s are formulas, then $(r \wedge s), (r \vee s), (\neg r)$ and $(r \rightarrow s)$ are formulas.	
	•	If x is a variable and p a formula, then $(\exists xp)$ and $(\forall xp)$ are formulas.	
Axioms:	•	proposition calculus axioms, specification axiom, "quantifier shift axiom"	
Rules of inference:	•	modus ponens, generalization rule	

Summary: Axioms & Rules of Inference

- Axioms:
  - Transfer of axioms from propositional calculus
  - Specification axiom: $(\forall x P(x) \rightarrow P(\mathit{value}))$, with *value* being a definite value (i. e., a substitution of a variable via a value)
  - "Quantifier shift axiom": $(\forall x(A \rightarrow B)) \equiv (A \rightarrow \forall x B)$, where x may not appear relevantly in A (i. e., moving the quantifier in)
- Rules of inference
  - Modus ponens – example from propositional calculus: A and $A \rightarrow B$ together imply B.
  - Generalization rule: A implies $(\forall x A)$

Rules

As we already discussed – rules are a classic and widely used representation for expert systems.

Example:

```
IF (battery OK)
AND (Value FuelGauge > 0)
AND (PetrolFilter clean)
THEN (Problem = IgnitionSystem)
```

Rules – Overview

Rules ...

- model human problem solving
- provide a rather natural representation of expert knowledge
- capture experiences based on problems which have been previously solved by the expert
- are adaptable regarding their basic formalism, to also incorporate e. g., statements about uncertainties or expectations
- usually offer a domain-specific compromise between expressive power and efficiency

Basic Types of Rules

- A rule consists of a *precondition* (premise) and an *action* (conclusion)
- Two types of actions:
  - Implications (w.r.t. truth values) $\rightsquigarrow$ (1) in the example
  - Changing a state ("side-effect") $\rightsquigarrow$ (2) in the example

Example:

(1)	If	1.	stiff neck and
		2.	high temperature and
		3.	impairment of consciousness occur together,
	then		meningitis is suspected.
(2)	stack(box1, box2)		
	if	1.	clear(box1)
		2.	holding(box2)
	then	1.	on(box2, box1)
		2.	clear(box2)
		3.	holding()

Inference – Forward Chaining

Components of the rule interpreter: 1. data base
2. rules

(1) DATA $\leftarrow$ initial data base,
(2) *until* DATA satisfies termination condition *do*
(3) *begin*
(4) choose executable rule R whose precondition is satisfied by DATA,
(5) DATA $\leftarrow$ result of applying action part of R to DATA,
(6) *end.*

Selection process (4) - two steps:

1. Pre-selection: Determine the set of all executable rules (*conflict set*), i. e., those which are currently applicable
2. Selection: Select a rule from the conflict set, using a conflict resolution strategy (trivial: select first rule; more advanced: apply most specific rule)

Frames / Semantic Networks

- Specifying properties of subjects
- Basically: *subject*, *predicate*, *object*, where *predicate* determines the type of the property, and *object* is the according property value.
- Alternatively: *individual*, *property*, *value*

As we have already seen:

- Logic (predicate):

$$prop(Individual, Property, Value)$$

- triple representation:

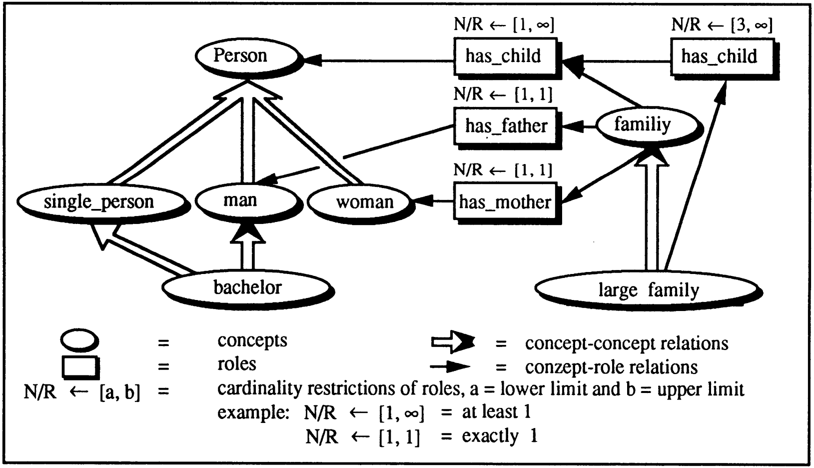
$$\langle individual, property, value \rangle$$

Frame Representations

- Basic idea: Set of facts can be better structured; properties of objects – providing stereotypes, with default values
- Similar to object-oriented formalisms (attributes, inheritance hierarchies, default procedures etc.)
- Inheritance: From general to more specific ($\rightsquigarrow$ reuse)
- Attached procedures – performed/monitor value changes

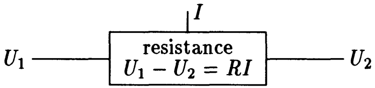
Object	Property	Values
elephant	is-a:	mammal
	color:	gray
	has:	trunk
	size:	large
	habitat:	ground

Example: Concepts *bachelor* & *large family* KL-ONE



Constraints

Example: Simulation of electrical circuits – Electrical Resistance



- Represent relationships between variables (undirected (!))
- In particular: Local boundary conditions, which need to be fulfilled for any possible solution
- Example – timetable/scheduling: One free day a week, for a particular person
- Solution space restricted by constraint
- Constraint system: Find solution, considering all restrictions
- Common forms: tables, functions, predicates, ...

Probabilistic Reasoning

- Classical logic: Proposition either true or false
- Uncertainty: Proposition with a certain probability
- Basic of probabilistic reasoning:
  - Attach probabilities to propositions
  - This expresses uncertainty
- Statistically derived uncertainties: probabilities
- Estimated by humans (difficult to provide exact probabilities): Uncertain reasoning $\rightsquigarrow$ evidence, certainty factors

Probabilistic Reasoning – Techniques

- Bayes' Theorem
- Historic systems: INTERNIST, MYCIN, MED1
- Dempster-Shafer Theorem
- Probabilistic Networks/Markov Networks
- Bayesian Networks

KR by Example

1. Diagnostic Scores Revisited
2. A Brief Perspective on Datalog
3. Short Intro to Description Logics

Diagnostic Scores Revisited I

For building a knowledge system using diagnostic scores, we first define inputs and outputs of such a system:

Definition 5 (Input and Output)

Let Ω_{sol} be the universe of all *solutions* (outputs) and Ω_a the set of all *attributes* (inputs). A range $dom(a)$ of values is assigned to each attribute $a \in \Omega_a$. Further, we assume Ω_{obs} to be the (universal) set of observable *input values* $a{:}v$, where $a \in \Omega_a$ is an attribute and $v \in dom(a)$ is an assignable value. An observable input $a{:}v$ is often called a *finding*. The universe set of all possible values is defined by Ω_V, i.e., $\Omega_V = \cup_{a \in \Omega_a} dom(a)$.

A diagnostic problem–solving session is described by a *case*.

Diagnostic Scores Revisited II

Definition 6 (Case)

A *case* c is defined as a tuple

$$c = (\, OBS_c, SOL_c \,)\,,$$

where $OBS_c \subseteq \Omega_{obs}$ is the *problem description* of the case, i.e., the observed finding set of the case c. The set $SOL_c \subseteq \Omega_{sol}$ is the set of solutions of the case.

Scoring rules are an intuitive representation for deriving solutions for a specified set of input values.

Diagnostic Scores Revisited III

Definition 7 (Scoring Rule)

A scoring rule r is denoted as follows

$$r = f_1 \diamond_1 \ldots \diamond_{n-1} f_n \xrightarrow{s} o \,,$$

where $f_i \in \Omega_{obs}$ are attribute values logically combined by conjunction or disjunction, i.e., $\diamond_i \in \{\wedge, \vee\}$, $s \in \boldsymbol{N}$ denotes the scoring point $sp(r)$ of the rule, and $o \in \Omega_{sol}$ is the targeted solution/output.

We call the collection of all scoring rules defined in a specific domain a *rule base* $\mathcal{R}$.

Diagnostic Score Pattern I

- When building a knowledge base using the Diagnostic Score pattern the domain specialist typically tries to rate all correlations between the findings and the solutions.
- Each finding–solution correlation is rated by a point score estimated by the domain specialist's experience.
- Using the representation of scoring rules, only rule conditions with a single finding are allowed to be defined in the knowledge base, i.e., $r = f_1 \diamond_1 \ldots \diamond_{n-1} f_n \xrightarrow{s} o$ with $n = 1$.
- By rating only single finding–solution relations by scores we avoid the creation of ambivalent or subsuming rules.

Diagnostic Score Pattern II

- In the pattern the scoring points of the particular rules are aggregated by a simple (linear) sum function.
- Thus, weightings of relations are not normalized, but the final state of a score is determined using a fixed threshold value.
- If the strength of a combination of attribute values is disproportionate when compared to the single observation of the attribute values, then the presented knowledge representation is not appropriate, since the particular attribute values can only contribute to a diagnostic score in a linear way.

Example: Diagnostic Score

Diagnostic Score table

	Duftlose Kamil...	Wiesen-Pippa...	Gem. Wegwart...
Blattform	Additional rules	Additional rules	
rundlich	N4	N3	N3
oval, eiförmig	N4		N2
oval, spitz zulaufend	N4		
länglich			P2
nadelartig			N5
röhrig (innen hohl)			N6
herzförmig	N6		N5
nierenförmig	N6		N5
pfeilförmig	$\boxed{\text{N6}}$		N2
gelappt	N5		
spatelig	N5	N3	N3
Blattbefestigung		Additional rules	Additional rules
gestielt	P2	N1	N2
geöhrt	N5		
stengelumfassend	N5	N2	
durchwachsen	N5	N5	N5
Stengelmilchsaft	Additional rules		Additional rules
ja, gelber Milchsaft		N5	
Stengelumfang	Additional rules	Additional rules	Additional rules
rund und eher glatt	N2	N1	N3
dreikantig	N3	N4	
vierkantig, quadratisch	N3	N4	
Stengelhaare			

Heuristic Decision Table Pattern

- Heuristic Decision Tables are a generalization of the Diagnostic Score pattern allowing more than one literal in the rule condition but *either* conjunctions *or* disjunctions in the rule condition.
- Often, the combination of findings can be used to directly derive a solution.
- Therefore, Heuristic Decision Tables typically do not make use of a complex score point schema but mostly contain rules immediately deriving a solution.

Example: Heuristic Decision Tree

Heuristic Decision table: Leaking air intake system

Leaking air intake system	R11 P7 and	R34 P4 and	R3 P4 and	R33 P3 and	R46 P3 or	R47 N4 and	R10 N7 and
Air intake system o.k.?							
YES							+
NO	+						
Known							
Unknown							
Mileage evaluation							
good mileage						+	
low mileage				+			
bad mileage		+					
Known							
Unknown							
Driving							
delayed take-off							
weak acceleration							
low idle speed							
insufficient power on partial lo					+	-	
insufficient power on full load					+	-	
unsteady idle speed			+				
no /else							
Known							
Unknown							

Datalog – Overview

- Declarative programming language (subset of Prolog)
- Grounded in logics: Subset of PL1 (function free Horn clauses)
- Often used as a query language in the context of *deductive databases*
- Many application areas:
  - Data integration
  - Information extraction
  - Machine learning
  - ...

Datalog

- Intuitively: Set of facts, rules and goals:
  - Facts $\rightsquigarrow$ "modeling relations"
  - Rules $\rightsquigarrow$ "infer new facts"
  - Goals $\rightsquigarrow$ "formalize queries"
- Example rule: $courseRS(T, S)$:- $course(C, T, S, P), professor(P, ``Sokrates'', R, Z), > (S, 2)$.
- Basic components: atoms/literals $q(A_1, \ldots, A_m)$, where q is the name of a predicate, or a built-in predicate (e.g., $<, =, >$). Example: $professor(S, ``Sokrates'', R, Z)$.
- Important: In Datalog, there are no functions (!)
- Datalog program as subset of predicate calculus formulas $\rightsquigarrow$ function free Horn clauses

Facts, Rules, and Goals

Literal: atom A oder negated atom $\texttt{not}(A)$

Fact: A

with atom A; e.g., `employee('John', 'B', 'Smith', ...)`

Rule: $\underbrace{A}_{\text{head}}\ \texttt{:-}\ \underbrace{B_1, \ldots, B_m}_{\text{body}}$

with atom A and literals B_i, example later

Goal: $\texttt{:-}\ B_1, \ldots, B_m$ or $\texttt{?-}\ B_1, \ldots, B_m$

with literals B_i

- A set of **facts** – modeling knowledge
- **Rules**: infer new knowledge
- **Goals**: used for expressing queries.

Datalog Rules – Notation

- $p(X_1, \ldots, X_m)$:- $q_1(A_{11}, \ldots, A_{1m_1}), \ldots, q_n(A_{n1}, \ldots, A_{nm_n})$
- Every $q_j(\ldots)$ is an atom (atomic formula); these q_j are often called *subgoals*
- $X_1, \ldots, X_m$ are *variables* - need to occur at least once on the right side of :-
- Logically equivalent form: $p(\ldots) \vee \neg q_1(\ldots) \vee \ldots \vee \neg q_n(\ldots)$
- This form is called a Horn clause

Example Datalog Program

... for identifying related pairs of courses ...:

- $\texttt{relCourse}(N_1, N_2)$:- $\texttt{require}(V, N_1)$, $\texttt{require}(V, N_2))$, $N_1 < N_2$.

- $\texttt{relTopics}(T_1, T_2)$:- $\texttt{relCourse}(N_1, N_2)$,
  $\texttt{course}(N_1, T_1, S_1, R_1)$, $\texttt{course}(N_2, T_2, S_2, R_2)$.

- $\texttt{buildsOn}(C, N)$:- $\texttt{require}(C, N)$.
  $\texttt{buildsOn}(C, N)$:- $\texttt{buildsOn}(C, M)$, $\texttt{require}(M, N)$.

- $\texttt{related}(N, M)$:- $\texttt{buildsOn}(N, M)$.
  $\texttt{related}(N, M)$:- $\texttt{buildsOn}(M, N)$.
  $\texttt{related}(N, M)$:- $\texttt{buildsOn}(V, N)$, $\texttt{buildsOn}(V, M)$.

Bottom–Up Evaluation of Datalog

- The bottom–up evaluation *iteratively* enlarges the relations for the predicates by repeatedly evaluating all rules until a fixpoint is reached (breadth–first).
- Every rule instance $A \text{ :– } B_1, \ldots, B_n$ derives facts A for its head predicate.
- The facts for the body predicates B_i are derived using rules themselves.
- Thus, it can happen that a rule transitively helps to derive facts for one of its body predicates (recursion).

Example

A DATALOG program containing the following 2 rules and 4 facts

```
a(X, Y) :- b(Y, X).
b(X, Y) :- c(X, Y), d(X).

a(5, 6).
c(1, 2).
c(3, 4).
d(3).
```

can be evaluated bottom–up by iteratively applying the rules and facts, respectively, in parallel as follows:

- The first iteration derives the 4 facts.
- The second iteration derives the 4 facts again plus (new):
  `b(3, 4).`
- The new fact (`b(3, 4).`) is derived using an instance of the second rule, whose body facts had been derived in iteration 1:

```
b(3, 4) :- c(3, 4), d(3).
```

Example (cont.)

- The first rule `a(X, Y) :- b(Y, X)` could not be used here, since we had no facts for `b` after iteration 1.
- The third iteration derives the 5 previously derived facts again plus the new fact

```
a(4, 3).
```

- The new fact is derived using an instance of the first rule, whose body fact had been derived in iteration 2:

```
a(4, 3) :- b(3, 4).
```

- Finally, no further new facts can be derived (fixpoint reached); we have obtained the following 6 facts:

```
a(5, 6).  a(4, 3).
b(3, 4).
c(1, 2).  c(3, 4).
d(3).
```

Short Intro to Description Logics

- Description logics (DLs) are one of the current KR paradigms
- Significant influence on standardization of Semantic Web languages (like OWL)
- Origin of DLs: semantic networks and frame-based systems
- DLs provide a formal semantics on logical grounds
- Can be seen as decidable fragments of first-order logic (PL1), closely related to modal logics
- Significant portion of DL-related research devoted to clarifying the computational effort of reasoning tasks in terms of their worst-case complexity
- Despite high complexity, even for expressive DLs there exist optimized reasoning algorithms with good average case behavior

DL Building Blocks

- Individuals: `martin`, `knowledge`, `systems`, etc.
  $\rightsquigarrow$ constants in PL1
- Concept names: `Person`, `Course`, `Student`, etc.
  $\rightsquigarrow$ unary predicates in PL1
- Role names: `hasFather`, `attends`, `worksWith`, etc.
  $\rightsquigarrow$ binary predicates in PL1
- can be subdivided into abstract and concrete roles (object und data properties)
- the set of all individual, concept and role names is called signature or vocabulary

Constituents of a DL Knowledge Base

TBox $\mathcal{T}$	information about concepts and their taxonomic dependencies
ABox $\mathcal{A}$	informationen about individuals, their concept and role memberships

in more expressive DLs also:

RBox $\mathcal{R}$	information about roles and their mutual dependencies

Example Description Logic

$\mathcal{ALC}$, Attribute Language with Complement, is the simplest DL that is Boolean closed

we define (complex) $\mathcal{ALC}$ concepts as follows:

- every **concept name** is a concept,
- $\top$ and $\bot$ are concepts,
- for concepts C and D, $\neg C$, $C \sqcap D$, and $C \sqcup D$ are concepts,
- for a role r and a concept$C$, $\exists r.C$ and $\forall r.C$ are concepts

Example: $\texttt{Student} \sqcap \forall \texttt{attendsCourse}.\texttt{MasterCourse}$
Intuitively: describes the concept comprising all students that attend only master courses

In our presentation of DL/ALC, we follow the structure/presentation according to [Hitzler et al. 2009]

Concept Axioms

For concepts C, D, a **general concept inclusion** (GCI) axiom has the form

$$C \sqsubseteq D$$

- $C \equiv D$ is an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$
- a **TBox** (terminological Box) consists of a set of GCIs

TBox $\mathcal{T}$

KB – ABox

an $\mathcal{ALC}$ ABox assertion can be of one of the following forms

- $C(a)$, called **concept assertion**
- $r(a, b)$, called **role assertion**

an ABox consists of a set of ABox assertions

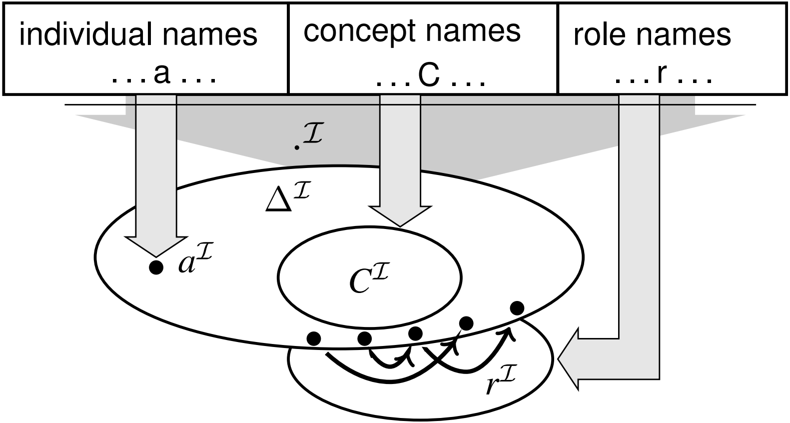
ABox $\mathcal{A}$

The Description Logic $\mathcal{ALC}$ – Semantics

Prototypical DL: Attributive Concept Language with Complements ($\mathcal{ALC}$)

- $\mathcal{ALC}$ is a syntactic variant of the modal logic **K**
- semantics defined in a model-theoretic way, that is, via interpretations
- can be expressed in first-order predicate logic
- a DL interpretation $\mathcal{I}$ consists of a domain $\Delta^{\mathcal{I}}$ and a function $\cdot^{\mathcal{I}}$, that maps
  - individual names a to domain elements $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
  - concept names C to sets of domain elements $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
  - role names r to sets of pairs of domain elements $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

$\mathcal{ALC}$: Schematic Representation – Interpretation



$\mathcal{ALC}$: Interpretation of Complex Concepts

The interpretation of complex concepts is defined inductively:

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
universal quantifier	$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}$ implies $y \in C^{\mathcal{I}}\}$
existential quantifier	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid$ there is some $y \in \Delta^{\mathcal{I}}$, such that $(x,y) \in r^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$

$\mathcal{ALC}$: Interpretation of Axioms

An interpretation can be extended to axioms:

name	syntax	semantic	notation
inclusion	$C \sqsubseteq D$	holds if $C^\mathcal{I} \subseteq D^\mathcal{I}$	$\mathcal{I} \models C \sqsubseteq D$
equivalence	$C \equiv D$	holds if $C^\mathcal{I} = D^\mathcal{I}$	$\mathcal{I} \models C \equiv D$
concept assertion	$C(a)$	holds if $a^\mathcal{I} \in C^\mathcal{I}$	$\mathcal{I} \models C(a)$
role assertion	$r(a, b)$	holds if $(a^\mathcal{I}, b^\mathcal{I}) \in r^\mathcal{I}$	$\mathcal{I} \models r(a, b)$

Logical Entailment in Knowledge Bases

- Let $\mathcal{I}$ be an interpretation, $\mathcal{T}$ a TBox, $\mathcal{A}$ an Abox and $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a knowledge base
- $\mathcal{I}$ is a **model for $\mathcal{T}$**, if $\mathcal{I} \models$ ax for every axiom ax in $\mathcal{T}$, written $\mathcal{I} \models \mathcal{T}$
- $\mathcal{I}$ is a **model for $\mathcal{A}$**, if $\mathcal{I} \models$ ax for every assertion ax in $\mathcal{A}$, written $\mathcal{I} \models \mathcal{A}$
- $\mathcal{I}$ is a **model for $\mathcal{K}$**, if $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$
- An axiom ax **follows** from $\mathcal{K}$, written $\mathcal{K} \models$ ax, if every model $\mathcal{I}$ of $\mathcal{K}$ is also a model of ax.

$\mathcal{ALC}$ Semantics via Translation into PL1

Translation of TBox axioms into PL1 through mapping π with C, D complex classes, r a role, and A an atomic class:

$$\pi(C \sqsubseteq D) = \forall x.(\pi_x(C) \rightarrow \pi_x(\mathrm{D})) \qquad \pi(C \equiv D) = \forall x.(\pi_x(C) \leftrightarrow \pi_x(\mathrm{D}))$$

$$\begin{aligned} \pi_x(A) &= A(x) & \pi_y(A) &= A(y) \\ \pi_x(\neg C) &= \neg \pi_x(C) & \pi_y(\neg C) &= \neg \pi_y(C) \\ \pi_x(C \sqcap D) &= \pi_x(C) \wedge \pi_x(\mathrm{D}) & \pi_y(C \sqcap D) &= \pi_y(C) \wedge \pi_y(\mathrm{D}) \\ \pi_x(C \sqcup D) &= \pi_x(C) \vee \pi_x(\mathrm{D}) & \pi_y(C \sqcup D) &= \pi_y(C) \vee \pi_y(\mathrm{D}) \\ \pi_x(\forall r.C) &= \forall y.(r(x,y) \rightarrow \pi_y(C)) & \pi_y(\forall r.C) &= \forall x.(r(y,x) \rightarrow \pi_x(C)) \\ \pi_x(\exists r.C) &= \exists y.(r(x,y) \wedge \pi_y(C)) & \pi_y(\exists r.C) &= \exists x.(r(y,x) \wedge \pi_x(C)) \end{aligned}$$

$\mathcal{ALC}$ – Example Knowledge Base

RBox $\mathcal{R}$

$$\texttt{own} \sqsubseteq \texttt{careFor}$$

"If somebody owns something, they care for it."

TBox $\mathcal{T}$

$$\texttt{Healthy} \sqsubseteq \neg\,\texttt{Dead}$$

"Healthy beings are not dead."

$$\texttt{Cat} \sqsubseteq \texttt{Dead} \sqcup \texttt{Alive}$$

"Every cat is dead or alive."

$$\texttt{HappyCatOwner} \sqsubseteq \exists\texttt{owns.Cat} \sqcap \forall\texttt{caresFor.Healthy}$$

"A happy cat owner owns a cat and everything he cares for is healthy."

ABox $\mathcal{A}$

$$\texttt{HappyCatOwner}\ (\texttt{schrödinger})$$

"Schrödinger is a happy cat owner. "

Summary

What Did We Learn?

- Overview: Intuitions on Inference and Knowledge
- Knowledge Engineering – Fundamental Principles
- First (Basic) Principles of Knowledge Representation
- Methods and Techniques for Knowledge Representation
- Knowledge Representation Exemplified – Three Examples