Chapter 1: Predicate Logic in Relational Databases And Beyond

Leopoldo Bertossi

Why Predicate Logic?

- Knowledge has to be represented in computers, and used by computers
- Formal logics, and predicate logic in particular, offer symbolic languages for that

  That can be processed symbolically as well
- Declarative specifications (as opposed to procedural, as with imperative programming languages)
- Automated theorem proving

  More generally: Automated Deduction
- Logic Programming: Prolog, Answer-Set Programming, Functional Programming, e.g. LISP, etc.
- In combination with probabilistic approaches: Probabilistic Logic Programming, Statistical Relational Learning, etc.

- They have been important in AI, and still are
- Alone and in combination with "neural" and "algorithmic" approaches to AI and ML: Neuro-Symbolic AI
- Predicate logic is at the basis of Relational Databases (RDBs)

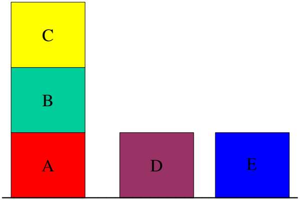
  Also part of the "tools" used -internally and externally- by Relational Database Systems
- Used in extensions of RDBs: Datalog, Ontologies, Knowledge Graphs, etc.
- Semantic Web
- Many other applications in Computer Science
- Predicate Logic is at the very origin of- and motivation for Computer Science

  In the mid 30s (Turing, etc.)

The Gist

- Description of this blocks world in predicate logic?

  A logical model of an outside reality?
- We want to say things like:
  - *"Every object that is on top of a block is not on the floor"*
  - *"C is a yellow block"*
  - *"C is on top of B"*
  - *"A is to the left of D"*
  - *"There is a blue block"*



- And define new or extend old properties:
  - *"A first object is to the left of a second object if it on top of a third object that is to the left of the second"*

  And be able to conclude (entail) that "B is to the left of D"

- We need a language that allows us to talk in general about properties (predicates) of individuals in our domain of discourse (outside reality)

  Not only about particular instantiations of properties, e.g. "C is on top of B"

  And to quantify over those individuals

- We will introduce such a language of predicate logic

  And present almost everything in the light of the example above

Languages of Predicate Logic

- More precisely: First-Order Predicate Logic (FOPL)
- The syntax (of languages) of FOPL offers a family of formal, symbolic languages

  Constructed according to a shared syntax (or grammar)

- First, introduce set $\mathcal{S}$ of symbols:
  1. Predicates (symbolic): $\mathit{On}(\cdot,\cdot)$, $\mathit{LeftOf}(\cdot,\cdot)$, $\mathit{Color}(\cdot,\cdot)$, $\mathit{Block}(\cdot)$ (each with a fixed arity, i.e. number of arguments)
  2. A special binary predicate for equality: $=$ (two arguments)
  3. Names to denote individuals: a, b, c, d, e, azul, rojo, ..., piso

     Also called "constants", they are used to name individuals from the external reality

     We are not forced to introduce names for every possible individual in the external reality

     There may be individuals -with or without a name in the domain- that do not have a name in the formal language

4. The typical propositional logical connectives: $\neg,\ \wedge, \vee,\ \rightarrow,\ \leftrightarrow$
5. Parenthesis: $), ($
6. Variables: $v, w, x, y, z, \ldots$ (an official and fixed list)

   They are intended to range over the individuals of a domain of discourse where the formal language is interpreted (later)
7. Quantifiers: $\forall,\ \exists$

- Now we can create some symbolic statements (sentences), among many others, that describe our blocks world:

  (official grammar coming)

  - $Block(a),\ On(b, a),\ LeftOf(a, d),\ \ a = a,\ \ldots$

    Atomic formulas, i.e. undecomposable into proper subformulas
  - More complex formulas:

    $\neg\ a = b, \quad (Block(a) \wedge On(b, a))$

    $\forall x \forall y \forall z((LeftOf(x, y) \wedge On(z, x)) \rightarrow LeftOf(z, y))$

    "for all ..."
  - $\forall x((\exists y Block(y) \wedge On(x, y)) \rightarrow \neg On(x, piso))$

    "for every object, if there is a block ...."

- Formulas are built with base alphabet in items 1. - 7.

  A.k.a. a "signature" ("schema" in RDBs)

- What are the precise grammar rules that allow to build official, legal formulas? Also part of the syntax

- Formulas are symbolic propositions defined by induction:

  1. Atomic formulas: Apply predicates to constants and variables

     $On(b, a), LeftOf(x, d), Block(c),\ Block(piso),\ On(azul, a),$ $y = rojo$

  2. If φ is a formula, then $\neg\varphi$ is a formula

     $\neg Color(a, verde),\ \ \neg\ a = rojo$ (abbreviated $a \neq rojo$)

  3. If φ, ψ are formulas, then propositional combinations of them are formulas, i.e. $(\varphi \wedge \psi),\ (\varphi \vee \psi),\ (\varphi \rightarrow \psi)$

     $(Block(a) \vee \neg On(z, a))$ (omit parenthesis when no danger of ambiguity)

  4. If φ is a formula and x is a variable, then $\forall x\varphi$ and $\exists x\varphi$ are formulas

     $\exists z(Block(a) \vee \neg On(z, a)),\ \ \forall z(Block(a) \vee \neg On(z, a))$

- (Legal or well-formed) formulas are obtained by applying a finite number of times the rules above
- $L(\mathcal{S})$ denotes the language (set of formulas) built on the basis of alphabet $\mathcal{S}$
- $\exists z(Block(a) \vee \neg(On(z,a) \wedge Color(a, verde)))$ is a formula?
  1. $Block(a)$, $On(z,a)$, $Color(a, verde)$ are atomic formulas by 1.
  2. $(On(z,a) \wedge Color(a, verde))$ is formula by 3.
  3. $\neg(On(z,a) \wedge Color(a, verde))$ is formula by 2.
  4. $(Block(a) \vee \neg(On(z,a) \wedge Color(a, verde)))$ is formula by 3.
  5. $\exists z(Block(a) \vee \neg(On(z,a) \wedge Color(a, verde)))$ is formula by 4.
- A generative grammar
- A program can check/generate legal formulas

  Recursion could be used

  Recursion and induction are the two sides of the same coin ...

- Among the formulas, the sentences do not have free variables
  They represent closed statements
- A free variable in a formula is one that appears outside the scope of a quantifier
  - $Color(a, rojo)$ trivially is sentence: no variables
  - $(On(z, a) \wedge Color(a, verde))$ not a sentence: variable z is free
  - $\exists z(On(z, a) \wedge Color(a, y))$ not a sentence: z is bound, but y is free
  - $Color(a, z) \wedge \exists z On(z, a)$ not a sentence: z appears both bound and free
  - $\forall y \exists z(On(z, a) \wedge Color(a, y))$ is a sentence
- Are those formulas above true?
- No idea, we only have the syntax
  No notion of meaning, interpretation, truth, logical consequence, ..., yet

Semantics of FOPL

- This is all symbolic so far
- What about meaning?

  What about truth of formulas?
- This is part of the semantics
- We need to interpret symbols, formulas Where?
- We need representations of "worlds", "realities", ..., where the symbolic elements can be interpreted
- We use semantic structures that stay in correspondence with the symbolic language
- They are abstract and simple representations in set-theoretic terms
- Structures are models of domain of discourse

  (Simplified) abstractions that capture the relevant aspects of the outside reality

- We know some of them from Mathematics

  Common numerical structures:

  $\mathfrak{R} = \langle \mathbb{R},\ =,\ <,\ +,\ \times,\ 0, 1 \rangle$ (ordered real numbers)

  $\mathfrak{N} = \langle \mathbb{N},\ =,\ <,\ +,\ \times,\ 0, 1 \rangle$

- Set-theoretic structures appear in Math: graphs, relations on sets, order relations on sets, vector spaces, groups, numerical structures, ...

- In more general terms, structures are of this form:

$$\mathfrak{S} = \langle U, R^U, \ldots, f^U, \ldots, c^U, \ldots \rangle$$

1. Domain (or universe): a non-empty set U
2. Relations between (domain) elements: R^U, ...

   Equality $=^U$ is always the "diagonal" of the domain, i.e.

   $=^U \ :=\ \{\langle u, u\rangle \mid u \in U\}$ (usually left implicit)
3. Functions/operations from U to U: f^U, ...
4. Distinguished elements of U: c^U, ...

Example: A structure $\mathfrak{B}$ representing our blocks world

- Domain $\mathcal{B} = \{A, B, C, D, E, green, yellow, gray, pink, purple, ..., floor, ...\}$
- Relations:

$$
\begin{aligned}
Block^{\mathcal{B}} &:= \{A, B, C, D, E\} \quad \text{(unary, i.e. } \subseteq \mathcal{B}\text{)} \\
On^{\mathcal{B}} &:= \{(A, floor), (B, A), (C, B), (D, floor), (E, floor)\} \\
&\qquad\qquad \text{(binary, } \subseteq \mathcal{B} \times \mathcal{B}\text{)} \\
Color^{\mathcal{B}} &:= \{(A, red), \ldots\} \\
LeftOf^{\mathcal{B}} &:= \{(A, D), (D, E)\} \\
=^{\mathcal{B}} &:= \{(A, A), (B, B), \ldots, (floor, floor)\} \quad \text{(usually left implicit)}
\end{aligned}
$$

- No functions or operations here
- Distinguished individuals: $A, B, C, D, E, green, ...$

  Our choice; no every element in the domain has to be distinguished (or has a name)

  So as with real numbers, most of them do not have a name or are distinguished
- $\mathfrak{B} = \langle \mathcal{B}, Block^{\mathcal{B}}, On^{\mathcal{B}}, Color^{\mathcal{B}}, LeftOf^{\mathcal{B}}, A, B, C, D, E, green, \ldots \rangle$

  is a structure

- This “finite” structure (the relations are finite) we basically have a ...

  A Relational Database!

$Block^{\mathcal{B}}$
A
B
$\dots$
E

$On^{\mathcal{B}}$	
A	$floor$
B	A
$\dots$	$\dots$
E	$floor$

$LeftOf^{\mathcal{B}}$	
A	D
D	E

- In RDBs the domain is not represented and is assumed to be infinite (the DB can be updated)
- In RDBs the equality is also left implicit (it is called a “built-in”, and queries can use it)
- Structure $\mathfrak{B}$ can be used to interpret language $L(S)$ we created (see page 6)
- Putting the formal language in correspondence with the structure