Defining New Predicates

- We could use FOPL to define new predicates

  This is what we do when we define views in RDBs

- Example: New predicate $ClearBlock(\cdot)$ "containing" blocks that are clear, with nothing on top

$$\forall x(\underbrace{ClearBlock(x)}_{\text{new}} \longleftrightarrow \underbrace{(Block(x) \land \neg\exists y\, On(y,x))}_{\text{already existing}}) \qquad (**)$$

- A new predicate symbol introduced in the language with its definition

  This formula can be stored in the DB and used in combination with the relations (tables)

  This predicate becomes a "virtual" one-argument relation

  Its contents can be computed upon request

- The new predicate can be used in new formulas

- Definitions in Math are of this kind

  They can be seen as convenient abbreviations/shorthands

  They could be eliminated, using the RHS instead

- Now we can obtain $\mathit{ClearBlock}(d)$ as an extra, possible virtual, ground atom

- The RHS of (**) can be seen as a query collecting all constants that satisfy its condition

  $\mathcal{Q}(x)\colon\ \mathit{Block}(x) \wedge \neg\exists y\, \mathit{On}(y,x)$

  Its answers become the contents of the new predicate

  $\mathit{Ans}(\mathcal{Q},\mathcal{D}) = \{c,d,e\}$ (see page 4)

- The query can be evaluated using the inductive/recursive definition of truth (more on this coming)

Defining Semantic Constraints

- We could also express semantic constraints (integrity constraints in RDBs)

- Example: *"Nothing can be on top of two different things"*

  $$\forall x \forall y \forall z (On(x, y) \wedge On(x, z) \;\rightarrow\; y = z) \qquad (***)$$

  A sentence we expect to see satisfied by $\mathcal{D}$

- A functional dependency: $On[1] \rightarrow On[2]$ (usual notation for FDs not an implication)

  The 2nd argument of On functionally depends upon the 1st

- *"Every object that is on top of something is a block"*

  $$\forall x \forall y (On(x, y) \;\rightarrow\; Block(x)) \qquad (4*)$$

  Equivalently: $\forall x (\exists y On(x, y) \;\rightarrow\; Block(x))$

  An inclusion or referential constraint

  In RDBs sometimes denoted $On[1] \subseteq Block[1]$

- ICs can be stored in the RDB
- Some may be automatically checked (verified) by the RDB
- Verifying an IC is like answering a Boolean query, i.e. that is *true* or *false*
- We can use (***) and (4*) together with the relations and views
- It is not only that we want them to be kept true by the RDB (IC maintenance)

  They can be put to good use:

  1. Metadata (i.e. data about the data)
  2. Additional synthetic knowledge
  3. Semantic query optimization, etc.

More on the FOPL/RDB Connection

- A RDB can be seen as a set-theoretic structure $\mathcal{D}$

  With a (possibly infinite) domain and certain finite relations defined on it

  Example:

Salaries	*Name*	*Salary*
	J. Page	*5,000*
	V. Smith	*3,000*
	M. Stowe	*7,000*
	K. Stein	*4,000*

Positions	*Name*	*Position*
	J. Page	*manager*
	V. Smith	*secretary*
	M. Stowe	*manager*
	K. Stein	*accountant*

- In RDBs the relations/tables are always finite

- The active domain is the finite set of elements of the database domain *Dom* that appear in the tables (*Dom* could be infinite)

  $ADom = \{J.Page, V.Smith, \ldots, 5{,}000, \ldots, accountant\} \subseteq Dom$

- A query $\mathcal{Q}(\bar{x})$ is a formula of FOPL with free variables $\bar{x}$
- The values for those variables when the database satisfies the query are the answers to the query from the database

Salaries	*Name*	*Salary*
	J. Page	*5,000*
	V. Smith	*3,000*
	M. Stowe	*7,000*
	K. Stein	*4,000*

Positions	*Name*	*Position*
	J. Page	*manager*
	V. Smith	*secretary*
	M. Stowe	*manager*
	K. Stein	*accountant*

Query: *"manager position with its salaries"*

$$\mathcal{Q}(x, y): \quad \exists z(Salaries(z, x) \ \wedge \ Positions(z, y) \ \wedge \ y = manager)$$

(join captured via variable in common z)

- In SQL:

```
SELECT  Salary, Position
FROM    Salaries, Positions
WHERE   Salaries.Name = Position.Name AND Position = 'manager'
```

(condition involves a join and a selection)

$$\mathcal{D} \models Q[5{,}000, \ manager] \qquad \mathcal{D} \models Q[7{,}000, \ manager]$$

(brackets contain the values for the variables that make the formula true, i.e. the answers from $\mathcal{D}$)

- The query is written in FOPL, also in its fragment called "Relational Calculus" (RC) (see below)

  In its RC and SQL versions it is a declarative query

  Saying *what* we want, not *how* to compute that

- It can be automatically and internally transformed into an imperative query in Relational Algebra (RA) (relying on the inductive structure of the query)

  The RA query evaluated as a sequence or relational/set operations on the DB

- The query on page 33 is of the most common kind of queries: conjunctive queries (CQs)

  In predicate logic (and relational calculus) they are expressed as $\exists$ followed by a conjunction ($\wedge$) of atomic formulas, including equality

  In RA they are of the form Project-Selection-Join (PSJ)

  No negation (or set difference) in them

- CQs are monotone: The set of answers from a DB can only grow when new atoms are inserted into the DB:

$$\mathcal{D} \subseteq \mathcal{D}' \implies \mathcal{Q}[\mathcal{D}] \subseteq \mathcal{Q}[\mathcal{D}']$$

(with $\mathcal{Q}[\mathcal{D}]$ denoting the set of answers to $\mathcal{Q}$ from $\mathcal{D}$)

On previous page: $\mathcal{Q}[\mathcal{D}] = \{(5{,}000, \ manager), \ (7{,}000, \ manager)\}$

- The query on page 33 can be evaluated or checked by inspecting only elements in *ADom* (as opposed to in *Dom*)

  That is, the query is *safe* or *domain-independent*

- "Query": $\mathcal{Q}(x, y){:} \ \neg Salaries(x, y)$

  It is legal formula of FOPL, but it is not safe as a query

  Evaluating it requires looking for values outside the tables

- It cannot be answered by a RDB

- In RDBs we only admit and use safe queries (ICs, view definitions, etc.)
  (it is good enough to inspect and search inside the tables)

- A query may have negation and still be safe

  "Employees who make more than 3K, but are not managers"

  $\mathcal{Q}(x)$: [redacted] $\exists z(\mathit{Salaries}(x,$ [redacted] $z) \land z > 3K \land \neg \mathit{Positions}(x, \mathit{manager}))$

- The Relational Calculus is the safe portion of FOPL

  In line with lack of the complement operation on sets

  Only set difference in RA

- Exercise: Relational schema:

  $\mathit{Frequents}(\mathit{Drinker}, \mathit{Bar})$, $\mathit{Serves}(\mathit{Bar}, \mathit{Beer})$, $\mathit{Likes}(\mathit{Drinker}, \mathit{Beer})$

  Express in relational calculus (FOPL) the query about the drinkers who do not frequent any bar that serves some beer they like

  Analyze (non-)monotonicity of the query

Discussion

- RDBs is one of the best known and most used forms of knowledge representation (KR)
- FOPL is at the very basis of them, and inspired their developments
- FOPL goes much beyond RDBs
- One can create Knowledge Bases (KBs) containing logical formulas for symbolic KR
- One can infer implicit knowledge from a KB by means of logical inference (deduction)

  Not only query answering (QA) as with RDBs
- QA can be seen as a very particular kind of inference
- We need to introduce some notions and techniques related to inference in FOPL (they apply to RDBs, but much beyond)

A Bit More on FOPL

- A sentence $\varphi \in L(\mathcal{S})$ of FOPL is universally valid if it is true under every interpretation structure $\mathcal{I}$ for $L(\mathcal{S})$
(of the general kind on page 12, in particular for RDBs)

  For every $\mathcal{I}$: $\mathcal{I} \models \varphi$ (think of the tautologies in propositional logic)

- Example: $\forall x(x = x)$ $\forall x P(x) \rightarrow P(c)$

  $P(c) \rightarrow \exists y P(y)$ (c a constant in $\mathcal{S}$)

  They symbolically capture the semantics (meaning) of equality and quantifiers

  FOPL versions of propositional tautologies, e.g.:
  $R(a, b) \vee \neg R(a, b)$

- Sentences $\varphi,\ \psi \in L(\mathcal{S})$ are logically equivalent, denoted $\varphi \equiv \psi$, iff $(\varphi \leftrightarrow \psi)$ is universally valid

  In every interpretation structure for the language they are simultaneously true or simultaneously false

  (a semantic notion; symbol "$\equiv$" belong to the metalanguage)

- Examples: These equivalences are easy to check

$$(\varphi \to \psi) \equiv (\neg\varphi \vee \psi) \qquad \neg\neg\varphi \equiv \varphi$$

$$\neg(R(a,b) \wedge S(b,c)) \equiv (\neg R(a,b) \vee \neg S(b,c))$$

$$\neg(R(a,b) \vee S(b,c)) \equiv (\neg R(a,b) \wedge \neg S(b,c))$$

$$\neg\forall x R(x) \equiv \exists x \neg R(x) \qquad \neg\exists x R(x) \equiv \forall x \neg R(x)$$

$$\exists x(R(x) \vee S(x)) \equiv (\exists x R(x) \vee \exists x S(x))$$

$$\forall x(R(x) \wedge S(x)) \equiv (\forall x R(x) \wedge \forall x S(x))$$

- However: $\forall x(R(x) \vee S(x)) \not\equiv (\forall x R(x) \vee \forall x S(x))$
  $\exists x(R(x) \wedge S(x)) \not\equiv (\exists x R(x) \wedge \exists x S(x))$

- To refute $\forall x(R(x) \vee S(x)) \equiv (\forall x R(x) \vee \forall x S(x))$ a counterexample suffices

  A structure that makes one true and the other false

  $\mathfrak{N} = \langle \mathbb{N}, \mathit{Even}, \mathit{Odd} \rangle$, with $\mathit{Even}, \mathit{Odd}$ the sets of even and odd numbers interpreting R, S, resp. makes the LHS true, but the RHS false

- Exercise: Prove that

  $\forall x \forall y (On(x,y) \rightarrow Block(x)) \equiv \forall x (\exists y On(x,y) \rightarrow Block(x))$

- Hint: One has to show that for an arbitrary structure $\mathcal{A} = \langle A, Block^A(\cdot), On^A(\cdot,\cdot)\rangle$ for the language at hand:

  (a) $\mathcal{A} \models \forall x \forall y (On(x,y) \rightarrow Block(x)) \implies$
  $\mathcal{A} \models \forall x (\exists y On(x,y) \rightarrow Block(x))$

  (b) $\mathcal{A} \models \forall x (\exists y On(x,y) \rightarrow Block(x)) \implies$
  $\mathcal{A} \models \forall x \forall y (On(x,y) \rightarrow Block(x))$

- Both implications are proved as usual in mathematics

- An interpretation structure $\mathcal{I}$ satisfies a set of sentences $\Sigma \subseteq L(\mathcal{S})$ iff it satisfies every sentence in Σ

$$\mathcal{I} \models \Sigma \quad :\Longleftrightarrow \quad \text{for every } \varphi \in \Sigma,\ \mathcal{I} \models \varphi \qquad (*)$$

- Example:

$\mathcal{I} \models \{\forall x(Ma(x) \rightarrow Mo(x)),\ \ Ma(c)\}$ iff

$\mathcal{I} \models \forall x(Ma(x) \rightarrow Mo(x))$ and $\mathcal{I} \models Ma(c)$ iff

$\mathcal{I} \models (\forall x(Ma(x) \rightarrow Mo(x)) \wedge Ma(c))$ (always possible when Σ finite)

- Set of sentences $\Sigma \subseteq L(S)$ and a sentence $\varphi \in L(S)$:

φ is a logical consequence of Σ iff φ is true in every interpretation structure that makes Σ true

$$\Sigma \models \varphi \quad :\Longleftrightarrow \quad \text{for every } \mathcal{I},\ \ \mathcal{I} \models \Sigma \ \Longrightarrow\ \mathcal{I} \models \varphi$$

(compare use of meta-symbol on LHS and (*); overloaded notation)

- Example: $\{\forall x(Ma(x) \rightarrow Mo(x)),\ \ Ma(c)\} \models Mo(c)$

- Notion of logical consequence is central in logic
- A semantic notion, because it is defined in terms of interpretation structures and truth
- Logical consequences is what we establish in mathematics when we prove that a theorem follows from a set of axioms (remember Geometry, Vector Spaces, etc.)
- We want to avoid obtaining logical consequences by appealing to interpretation structures

  Want to avoid reasoning at the meta-level
- We need a purely symbolic, formal, deductive, mechanical version/counterpart of logical consequence
- Something that can be automated as pure symbolic processing of formulas

- We provide such symbolic deductive process for formulas of a particular, but important, syntactic form

  Via examples ...

- We consider clauses, which are disjunctions of literals, i.e. of atomic or negations of atomic formulas

- Examples: These are clauses: $R(a, b)$, $\neg S(a, b)$, $R(a, b) \vee \neg S(a, b)$, $P(x) \vee \neg R(x, a) \vee U(y, x)$

  In clauses the variables are implicitly universally quantified

  $P(x) \vee \neg R(x, a) \vee U(y, x)$ is indeed $\forall x \forall y (P(x) \vee \neg R(x, a) \vee U(y, x))$

- The resolution deduction rule works with clauses

  It takes two clauses and produces a new clause by eliminating "complementary" literals after the initial clauses have been unified

- There is a symbolic method to translate arbitrary formulas into sets of clauses

  For example, $\forall x (Ma(x) \rightarrow Mo(x))$ is logically equivalent to the clause $\neg Ma(x) \lor Mo(x)$ (see page 39)

- Example:

$$\frac{\begin{array}{c} \neg \cancel{Ma}(x) \lor Mo(x) \\ \cancel{Ma}(c) \end{array}}{Mo(c)}$$

  After unifying $Ma(x)$ with $Ma(c)$ via $x := c$

- The "resolvent" (bottom) is logical consequence of the "parent" clauses (above)

- Purely syntactical, symbolic, logical "calculus"

  Hence "calculus" in RC

- Example: Our knowledge base is the set of formulas:

$$\Sigma = \{On(c,b), On(b,a), LeftOf(a,d),$$
$$\forall x \forall y \forall z (LeftOf(x,y) \wedge On(z,x) \rightarrow LeftOf(z,y))\}$$

- We want to conclude that: $\Sigma \models LeftOf(c,d)$

1. Last formula in Σ is equivalent to the (implicitly quantified):

   $\neg(LeftOf(x,y) \wedge On(z,x)) \vee LeftOf(z,y)$

   $\neg\cancel{L}eftOf(x,y) \vee \neg On(z,x) \vee LeftOf(z,y)$ (a clause)

2. $\cancel{L}eftOf(a,d)$ (from Σ)
3. $\neg\cancel{O}n(z,a) \vee LeftOf(z,d)$ (resolvent)
4. $\cancel{O}n(b,a)$ (from Σ)
5. $\cancel{L}eftOf(b,d)$ (resolvent)
6. $\neg\cancel{L}eftOf(x,y) \vee \neg On(z,x) \vee LeftOf(z,y)$
7. $\neg\cancel{O}n(z,b) \vee LeftOf(z,d)$ (resolvent)
8. $\cancel{O}n(c,b)$ (from Σ)
9. $LeftOf(c,d)$ (resolvent)

- This form of purely symbolic deduction has been implemented in many computational systems

  It is at the basis of the PROLOG programming language (for PROgramming in LOGic)

  And deductive extensions of relational DBs (RDBs)

  And automated theorem provers (OTTER, PROVER9, VAMPIRE, ...)

- Resolution + Unification was an important step in AI ($\sim$ 1965)

- Any formula can be transformed into a set of clauses

  The issue are the existential quantifiers (clauses do not have them)

  There is a “trick” ...

- Examples:

  $\exists x \forall y (Block(y) \to \neg On(y,x)) \mapsto \exists x \forall y (\neg Block(y) \lor \neg On(y,x)) \mapsto$

  $\forall y (\neg Block(y) \lor \neg On(y,c)) \mapsto \neg Block(y) \lor \neg On(y,c)$ (c fresh constant)

  $\forall y \exists x (Block(y) \to \neg On(y,x)) \mapsto \forall y \exists x (\neg Block(y) \lor \neg On(y,x)) \mapsto$

  $\forall y (\neg Block(y) \lor \neg On(y,f(y))) \mapsto \neg Block(y) \lor \neg On(y,f(y))$

  (f a fresh function symbol)

- The logical equivalences on page 39 are useful to obtain clauses Two additional useful ones:

  $\exists x(\varphi \land \psi(x)) \equiv (\varphi \land \exists x \psi(x))$ and $\forall x(\varphi \lor \psi(x)) \equiv (\varphi \lor \forall x \psi(x))$

  When x does not appear free in ψ

  Example: $\exists y \forall x (\forall u Q(x,u,z) \to \forall z P(y,u,z)) \equiv$
  $\exists y \forall x (\neg \forall u Q(x,u,z) \lor \forall z P(y,u,z)) \equiv$
  $\exists y \forall x (\exists u \neg Q(x,u,z) \lor \forall z P(y,u,z)) \equiv$
  $\exists y \forall x (\exists v \neg Q(x,v,z) \lor \forall z P(y,u,z)) \equiv$ (!)
  $\exists y \forall x \exists v (\neg Q(x,v,z) \lor \forall z P(y,u,z)) \equiv$
  $\exists y \forall x \exists v (\neg Q(x,v,z) \lor \forall w P(y,u,w)) \equiv$

  $\exists y \forall x \exists v \forall w (\neg Q(x,v,z) \lor P(y,u,w))$ (not a clause yet)

- What do we do with this?

$$\exists y \forall x \exists v \forall w (\neg Q(x, v, z) \vee P(y, u, w)) \qquad (*)$$

A prefix of quantifiers; two of them existential; the rest is fine ...

- Now we use a Skolem constant c for y, and a Skolem function $f(x)$ for z: $\forall x \forall w (\neg Q(x, f(x), z) \vee P(c, u, w))$

$$\neg Q(x, f(x), z) \vee P(c, u, w)) \qquad \text{(a clause)}$$

- This clause is not logically equivalent to (*)

  They do not even share the same language

- One can prove that the original FOPL KB and that with the computed clauses are equiconsistent: One is consistent iff the other is consistent

  KB Σ is consistent if there is an interpretation $\mathcal{I}$ making it true ($\mathcal{I} \models \Sigma$)

  This is good enough for the form in which we use resolution most of the time (a bit of this later)

Discussion

- FOPL is relevant in KR and other areas of AI

  It will keep reappearing in different forms and contexts later on

- Using KBs written in full FOPL is perfectly fine, but reasoning may be computationally very expensive (undecidable, uncomputable)

- One commonly uses better behaved fragments of FOPL for KR

  Syntactic subclasses of formulas of a FOPL language $L(\mathcal{S})$

- In the next chapter we will do this in the context of RDBs

- We will extend RC as a query language into a more expressive one (Datalog, ...), but still computationally manageable

  At the same time, the extension will be syntactically restricted

  Both a restriction and an extension of FOPL for RDBs

- Those extensions can be used for KR in general