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(x)` “containing” blocks that are clear, with nothing on top.

\[
\forall x (\text{ClearBlock}(x) \leftrightarrow (\text{Block}(x) \land \neg \exists y \text{On}(y, x)))
\]  

- 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).

- The new predicate can be used in new formulas.

- This predicate becomes a “virtual” one-argument relation. Its contents can be computed upon request.
• 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 \textit{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
  \[ Q(x) : \ Block(x) \land \neg \exists y \ On(y, x) \]
  Its answers become the contents of the new predicate
  \[ \text{Ans}(Q, D) = \{c, d, e\} \] (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”

\[
\forall x \forall y \forall z \left( On(x, y) \land On(x, z) \rightarrow y = z \right) \quad (***)
\]

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”

\[
\forall x \forall y \left( On(x, y) \rightarrow Block(x) \right) \quad (4*)
\]

Equivalently:

\[
\forall x \left( \exists y \ On(x, y) \rightarrow Block(x) \right)
\]

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:

<table>
<thead>
<tr>
<th>Salaries</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Page</td>
<td>5,000</td>
<td></td>
</tr>
<tr>
<td>V. Smith</td>
<td>3,000</td>
<td></td>
</tr>
<tr>
<td>M. Stowe</td>
<td>7,000</td>
<td></td>
</tr>
<tr>
<td>K. Stein</td>
<td>4,000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Positions</th>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Page</td>
<td>manager</td>
<td></td>
</tr>
<tr>
<td>V. Smith</td>
<td>secretary</td>
<td></td>
</tr>
<tr>
<td>M. Stowe</td>
<td>manager</td>
<td></td>
</tr>
<tr>
<td>K. Stein</td>
<td>accountant</td>
<td></td>
</tr>
</tbody>
</table>

- In RDBs the relations/tables are always finite

- The active domain is the finite set of elements of the database domain $\text{Dom}$ that appear in the tables ($\text{Dom}$ could be infinite)

$\mathcal{A}\text{Dom} = \{ \text{J. Page}, \text{V. Smith}, \ldots, 5,000, \ldots, \text{accountant} \} \subseteq \text{Dom}$
• A query $Q(\bar{x})$ 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

<table>
<thead>
<tr>
<th>Salaries</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Page</td>
<td>5,000</td>
<td></td>
</tr>
<tr>
<td>V. Smith</td>
<td>3,000</td>
<td></td>
</tr>
<tr>
<td>M. Stowe</td>
<td>7,000</td>
<td></td>
</tr>
<tr>
<td>K. Stein</td>
<td>4,000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Positions</th>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>J. Page</td>
<td>manager</td>
<td></td>
</tr>
<tr>
<td>V. Smith</td>
<td>secretary</td>
<td></td>
</tr>
<tr>
<td>M. Stowe</td>
<td>manager</td>
<td></td>
</tr>
<tr>
<td>K. Stein</td>
<td>accountant</td>
<td></td>
</tr>
</tbody>
</table>

Query: “manager position with its salaries”

$Q(x, y) : \exists z (\text{Salaries}(z, x) \land \text{Positions}(z, y) \land y = \text{manager})$

(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)

$D \models Q[5,000, \text{manager}]$ \quad $D \models Q[7,000, \text{manager}]$

(brackets contain the values for the variables that make the formula true, i.e. the answers from $D$)
• 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 ∃ followed by a conjunction (∧) 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:

\[ D \subseteq D' \implies Q[D] \subseteq Q[D'] \]

(with \( Q[D] \) denoting the set of answers to \( Q \) from \( D \))

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

• The query on page 33 can be evaluated or checked by inspecting only elements in \( A\text{Dom} \) (as opposed to in \( \text{Dom} \))

That is, the query is safe or domain-independent

• “Query”: \( Q(x, y): \neg \text{Salaries}(x, y) \)

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”

$$Q(x) : \exists y \exists z (Salaries(x, z) \land z > 3K \land \neg Positions(x, manager))$$

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:

$$Frequents(Drinker, Bar), Serves(Bar, Beer), Likes(Drinker, Beer)$$

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 \( \varphi \in L(S) \) of FOPL is **universally valid** if it is true under every interpretation structure \( \mathcal{I} \) for \( L(S) \)

  (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:**
  \[
  \forall x(x = x) \quad \forall xP(x) \rightarrow P(c) \\
  P(c) \rightarrow \exists yP(y) \quad (c \text{ a constant in } S)
  \]

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

  FOPL versions of propositional tautologies, e.g.:
  \[
  R(a, b) \lor \neg R(a, b)
  \]

- Sentences \( \varphi, \psi \in L(S) \) 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

\[(\varphi \to \psi) \equiv (\neg \varphi \lor \psi) \quad \neg \neg \varphi \equiv \varphi\]

\[\neg (R(a, b) \land S(b, c)) \equiv (\neg R(a, b) \lor \neg S(b, c))\]

\[\neg (R(a, b) \lor S(b, c)) \equiv (\neg R(a, b) \land \neg S(b, c))\]

\[\neg \forall x R(x) \equiv \exists x \neg R(x) \quad \neg \exists x R(x) \equiv \forall x \neg R(x)\]

\[\exists x (R(x) \lor S(x)) \equiv (\exists x R(x) \lor \exists x S(x))\]

\[\forall x (R(x) \land S(x)) \equiv (\forall x R(x) \land \forall x S(x))\]

• However:

\[\forall x (R(x) \lor S(x)) \not\equiv (\forall x R(x) \lor \forall x S(x))\]

\[\exists x (R(x) \land S(x)) \not\equiv (\exists x R(x) \land \exists x S(x))\]

• To refute \[\forall x (R(x) \lor S(x)) \equiv (\forall x R(x) \lor \forall x S(x))\] a counterexample suffices

A structure that makes one true and the other false

\[\mathcal{N} = \langle \mathbb{N}, \text{Even}, \text{Odd} \rangle, \text{ with Even, Odd the sets of even and odd numbers interpreting } R, S, \text{ resp. makes the LHS true, but the RHS false}\]
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 \( A = \langle A, Block^A(\cdot), On^A(\cdot, \cdot) \rangle \) for the language at hand:

(a) \( A \models \forall x \forall y (On(x, y) \rightarrow Block(x)) \implies A \models \forall x (\exists y On(x, y) \rightarrow Block(x)) \)

(b) \( A \models \forall x (\exists y On(x, y) \rightarrow Block(x)) \implies A \models \forall x \forall y (On(x, y) \rightarrow Block(x)) \)

Both implications are proved as usual in mathematics
• An interpretation structure $\mathcal{I}$ satisfies a set of sentences $\Sigma \subseteq L(S)$ iff it satisfies every sentence in $\Sigma$

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

• Example:

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

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

$\mathcal{I} \models (\forall x (Ma(x) \to Mo(x)) \land Ma(c))$ (always possible when $\Sigma$ finite)

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

$\varphi$ is a logical consequence of $\Sigma$ iff $\varphi$ is true in every interpretation structure that makes $\Sigma$ true

$$\Sigma \models \varphi \quad \iff \text{for every } \mathcal{I}, \quad \mathcal{I} \models \Sigma \implies \mathcal{I} \models \varphi$$

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

• Example: $\{\forall x (Ma(x) \to Mo(x)), \ Ma(c)\} \models Mo(c)$
• 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: \( R(a, b), \neg S(a, b), \)
  \( R(a, b) \lor \neg S(a, b), \quad P(x) \lor \neg R(x, a) \lor U(y, x) \)

In clauses the variables are implicitly universally quantified
\( P(x) \lor \neg R(x, a) \lor U(y, x) \) is indeed \( \forall x \forall y (P(x) \lor \neg R(x, a) \lor U(y, x)) \)

• 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:

\[
\begin{array}{c}
\neg Ma(x) \lor Mo(x) \\
Ma(c) \\
\hline
Mo(c)
\end{array}
\]

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:

\[ \Sigma = \{ \text{On}(c, b), \text{On}(b, a), \text{LeftOf}(a, d), \]
\[ \forall x \forall y \forall z (\text{LeftOf}(x, y) \land \text{On}(z, x) \rightarrow \text{LeftOf}(z, y)) \}\]

• We want to conclude that: \( \Sigma \models \text{LeftOf}(c, d) \)

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

\[ \neg(\text{LeftOf}(x, y) \land \text{On}(z, x)) \lor \text{LeftOf}(z, y) \]
\[ \neg \text{LeftOf}(x, y) \lor \neg \text{On}(z, x) \lor \text{LeftOf}(z, y) \quad \text{(a clause)} \]

2. \( \text{LeftOf}(a, d) \) (from \( \Sigma \))

3. \( \neg \text{On}(z, a) \lor \text{LeftOf}(z, d) \) (resolvent)

4. \( \text{On}(b, a) \) (from \( \Sigma \))

5. \( \text{LeftOf}(b, d) \) (resolvent)

6. \( \neg \text{LeftOf}(x, y) \lor \neg \text{On}(z, x) \lor \text{LeftOf}(z, y) \)

7. \( \neg \text{On}(z, b) \lor \text{LeftOf}(z, d) \) (resolvent)

8. \( \text{On}(c, b) \) (from \( \Sigma \))

9. \( \text{LeftOf}(c, d) \) (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 (∼ 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:

\[ \exists x \forall y (\text{Block}(y) \rightarrow \neg \text{On}(y, x)) \leftrightarrow \exists x \forall y (\neg \text{Block}(y) \lor \neg \text{On}(y, x)) \]

\[ \forall y (\neg \text{Block}(y) \lor \neg \text{On}(y, c)) \leftrightarrow \neg \text{Block}(y) \lor \neg \text{On}(y, c) \quad (c \text{ fresh constant}) \]

\[ \forall y \exists x (\text{Block}(y) \rightarrow \neg \text{On}(y, x)) \leftrightarrow \forall y \exists x (\neg \text{Block}(y) \lor \neg \text{On}(y, x)) \leftrightarrow \]

\[ \forall y (\neg \text{Block}(y) \lor \neg \text{On}(y, f(y))) \leftrightarrow \neg \text{Block}(y) \lor \neg \text{On}(y, f(y)) \quad (f \text{ a fresh function symbol}) \]

The logical equivalences on page 39 are useful to obtain clauses

Two additional useful ones:

\[ \exists x (\varphi \land \psi(x)) \equiv (\varphi \land \exists x \psi(x)) \quad \text{and} \quad \forall x (\varphi \lor \psi(x)) \equiv (\varphi \lor \forall x \psi(x)) \]

When \( x \) does not appear free in \( \psi \)

Example:

\[ \exists y \forall x (\forall u Q(x, u, z) \rightarrow \forall z P(y, u, z)) \equiv \]

\[ \exists y \forall x (\neg \forall u Q(x, u, z) \lor \forall z P(y, u, z)) \equiv \]

\[ \exists y \forall x (\exists u \neg Q(x, u, z) \lor \forall z P(y, u, z)) \equiv \]

\[ \exists y \forall x (\exists v \neg Q(x, v, z) \lor \forall z P(y, u, z)) \equiv \]

\[ \exists y \forall x \exists v (\neg Q(x, v, z) \lor \forall z P(y, u, z)) \equiv \]

\[ \exists y \forall x \exists v (\neg Q(x, v, z) \lor \forall w P(y, u, w)) \equiv \]

\[ \exists y \forall x \exists v \forall w (\neg Q(x, v, z) \lor P(y, u, w)) \quad \text{(not a clause yet)} \]
• What do we do with this?

\[ \exists y \forall x \exists v \forall w (\neg Q(x, v, z) \lor P(y, u, w)) \]  

(*)

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 \( z \):

\[ \forall x \forall w (\neg Q(x, f(x), z) \lor P(c, u, w)) \]

\[ \neg Q(x, f(x), z) \lor P(c, u, w) \]  

(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 \( \Sigma \) 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 \( L(S) \)
- 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