Chapter 2: Datalog and Deductive Extensions of Relational Databases

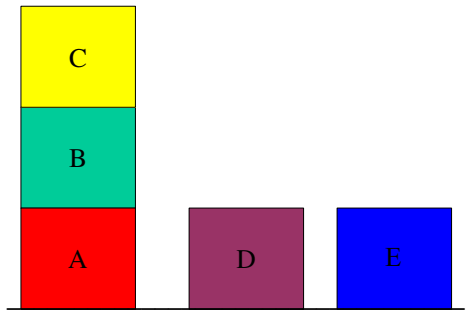
Leopoldo Bertossi

The Need for Extensions of RC

- Back to the blocks world
- Want to define the predicate (view) $Above(\cdot, \cdot)$

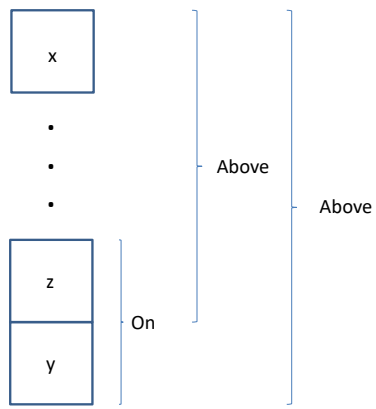
  True when an object is above another on the same stack, maybe with other objects in between

  For example, block C should be above block A



- Can we use RC to define this predicate?
- We want a definition that does not depend on this specific configuration

  It should work if we change this world (or DB), e.g. adding blocks



- A first attempt ...

- What about:

$$\forall x \forall y (On(x, y) \rightarrow Above(x, y)) \ \wedge$$
$$\forall x \forall y (\exists z (Above(x, z) \wedge On(z, y)) \ \rightarrow \ Above(x, y)) \qquad (*)$$

- A first "objection": Not of the form of an explicit definition

  With the new one side of a "$\leftrightarrow$" or an "$=$", and something known on the other

- However, we do obtain $Above(c, a)$: Use first conjunct above

  From $On(c, b) \rightarrow Above(c, b)$ plus $On(c, b)$, obtain $Above(c, b)$

  Next, use second conjunct above

  From $\underbrace{\exists z (Above(c, z) \wedge On(z, a))}_{z=b} \ \rightarrow \ Above(c, a)$ plus

  $Above(c, b)$ and $On(b, a)$: obtain $Above(c, a)$

- $\forall x \forall y (On(x,y) \rightarrow Above(x,y)) \ \wedge$

  $\forall x \forall y (\exists z (Above(x,z) \wedge On(z,y)) \ \rightarrow \ Above(x,y))$ (*)

- A "glitch": $Above(d,a)$ can be true in the light of (*), and it shouldn't

  That is, there are structures where $Above(d,a)$ and the formulas in (*) are both true

  For example, structures where $Above(d,a)$ is true, and the antecedents of the implications in (*) are false (with which the implications become true; check this!)

- What about

  $\forall x \forall y (\underbrace{Above(x,y)}_{\text{the new}} :\longleftrightarrow \ (\underbrace{On(x,y)}_{\text{base case}} \vee \exists z (\underbrace{Above(x,z)}_{\text{recursive "call"}} \wedge On(z,y)))$?

  A recursive definition?

- It does not work! For the same reason as above

  Cannot force $Above$ to contain exactly what is depicted on page 2

- Are we not clever enough to properly use RC (or FOPL)?

  No!

- In RC (or FOPL) it is mathematically impossible to define the transitive closure (TC) of a binary relation!

  The smallest transitive binary relation that includes a given one (under set inclusion)

  And this is what we are trying to do here ...

- The required predicate minimality cannot be expressed

- The same applies to Relational Algebra (RA)

  RA has the same expressive power as RC

- You may remember from "Discrete Math" that an algebraic iteration is needed for the TC, whose number of steps depends on the initial relation

- This issue already appears in usual RDBs
- If one has a table showing direct flight connections, and one tries to fly between cities using connections

  Those indirect routes are not stored in the DB, and the connections have to be computed
- One flies by transitive closure ...
- An stored iterative procedure will compute the routes

  This will go beyond pure RA, but will use relational operations as intermediate steps
- Or one can use something like Datalog, as to be seen ...

Datalog

- Datalog is a logical language that extends (part of) RC with recursion
- Used as a query language, and for view definitions on RDBs

  As a "Datalog program"
- Proposed and investigated in the mid 80s
- Constructs of Datalog found its way into commercial relational DBMSs
- After some dormant period, it is back, healthy and strong

  As the basis for many applications inside and outside RDBs
- Can be seen as enabling a "deductive" extension of RDBS: Deductive DBs!
- Several "ontological" extensions of Datalog (will come back)
- A Datalog program extends a RDB represented as a Herbrand structure

Example: DB $\mathcal{D}$

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

- A Datalog program defining a view:

  $$Top(x) \longleftarrow Salaries(x, y),\ y > 3{,}000 \qquad (*)$$

  (variables universally quantified; comma stands for $\wedge$)

- This is a "rule" defining the predicate on the LHS

  It can be written as a clause:

  $$\forall x \forall y (Top(x) \vee \neg Salaries(x, y) \vee \neg\ y > 3{,}000)$$

  Equivalent to: $\forall x (Top(x) \vee \forall y (\neg Salaries(x, y) \vee \neg\ y > 3{,}000))$

  So, think of (*) as: $Top(x) \longleftarrow \exists y (Salaries(x, y) \wedge\ y > 3{,}000)$

- Compute extension (contents) of predicate *Top* by going to the RHS and collecting what is true, and nothing more:

  $$Top[\mathcal{D}] = \{\langle J.Page\rangle,\ \langle M.Stowe\rangle, \langle K.Stein\rangle\}$$

- A form of minimization (extension of CWA to be made precise later on)

  For *V.Smith* the implication is true, because the body is false; but being the body false, it is not collected

- For *Top* we obtain the same as in Chapter 1 with RC or RA

  There is nothing like minimization needed in this case

- We would expect Datalog to work fine when defining the TC

- Notice that the program and the RDB belong to the same level of language, the object or symbolic language

  For this reason it is common to list the contents of the DB as a set of facts (atomic formulas w/o free variables) in the program

$$\begin{aligned} Top(x) &\longleftarrow Salaries(x, y),\ y > 3{,}000 \\ Salaries(J.Page, 5, 000) &\longleftarrow \\ &\ldots \end{aligned}$$

  Symbol $\leftarrow$ on the RHS of facts usually omitted; they are true without conditions

- The program can be seen as an extension of the DB (now as set of ground facts), which called the extensional database (EDB) of the program

  The rules form the intentional database (IDB), and can be seen as a set of view definitions or queries

Example: Relational DB $\mathcal{D} = \{Arc(b, c), Path(b, b), Path(c, c)\}$

Datalog program Π on top of $\mathcal{D}$:

$$Path(x, z) \leftarrow Arc(x, y),\ Path(y, z) \qquad (*)$$

- A recursive definition of *Path* (or extension of the partial definition already in the DB)
- A predicate defined in terms of itself!

  Paths defined in terms of shorter paths
- Now we give a precise, "model-theoretic" or "model-based" semantics to Datalog programs (using this example)
- In terms of the intended models of the specification (program)
- By definition, what is true wrt. the program is what is true in the intended models of the program

  This idea can be applied to all kinds of logic-based specifications

- What is the semantics of Π? (its meaning)
- What world is Π describing?

  Is there an intended model (the world) for Π?
- We concentrate on Herbrand structures

  Those built with the Herbrand universe $H = \{b, c\}$
- Now, with the program's predicates we consider the Herbrand Base (HB) of Π

$$HB(\Pi) = \{Arc(b,b), Arc(c,c), Arc(b,c), Arc(c,b), \\ Path(b,b), Path(c,c), Path(b,c), Path(c,b)\}$$

  It contains all the possible ground atomic formulas that can be built with the program's language
- Each of the 2^8 subsets of $HB(\Pi)$ will be a Herbrand structure

  Each of them looking like the DBs (c.f. Chapter 1)

- Some Herbrand structures will make the program true, others not ...

  Those that do, will be the Herbrand models of the program (just "models" from now on)

- Making the program true means:
  - Making all the rules true, as usual implications
  - In particular, they have to contain all the facts of the program, i.e. they have to contain $\mathcal{D}$

- For the first item, we consider (at least conceptually) all the ground instantiations on H of the program's rules

$$
\begin{array}{rcl}
Path(b,b) & \leftarrow & Arc(b,b),\ Path(b,b) \\
Path(b,c) & \leftarrow & Arc(b,b),\ Path(b,c) \\
\ldots & \leftarrow & \ldots \\
Path(b,b) & \leftarrow & \\
Path(c,c) & \leftarrow & \\
Arc(b,c) & \leftarrow &
\end{array}
$$

$$
\begin{array}{rcl}
Path(b,b) & \leftarrow & Arc(b,b),\ Path(b,b) \\
Path(b,c) & \leftarrow & Arc(b,b),\ Path(b,c) \\
\ldots & \leftarrow & \ldots \\
Path(b,b) & \leftarrow & \\
Path(c,c) & \leftarrow & \\
Arc(b,c) & \leftarrow &
\end{array}
$$

- Take $\mathcal{M}_1 = \{Arc(b,c), Path(b,b), Path(c,c), Path(b,c)\}$, a candidate to be a model
- Ground rule $Path(b,c) \leftarrow Arc(b,c),\ Path(c,c)$ is true in $\mathcal{M}_1$ iff when all the atoms in the body (RHS) are true, i.e. belong to $\mathcal{M}_1$, then also the head (LHS) belongs to $\mathcal{M}_1$

  $\mathcal{M}_1$ makes this rule true:

  $$\mathcal{M}_1 \models (Path(b,c) \leftarrow Arc(b,c),\ Path(c,c))$$

- Also: $\mathcal{M}_1 \models (Path(b,c) \leftarrow Arc(b,b),\ Path(b,c))$

  Trivially, because the body is false: $Arc(b,b) \notin \mathcal{M}_1$

- It turns out that $\mathcal{M}_1 \models \Pi$, i.e. it is a model of the program

  It makes all ground rules true

- Another model of the program: (check it!)

  $\mathcal{M}_2 = \{Arc(b,b), Arc(c,c), Arc(b,c), Arc(c,b), Path(b,b), Path(c,c), Path(b,c), Path(c,b)\}$

  Actually, the Herbrand base itself; always a model ...

- Yet another model

  $\mathcal{M}_3 = \{Arc(b,c), Path(b,b), Path(c,c), Path(b,c), Path(c,b)\}$

  Last atom not "justified" by rule (*), but still the implication is true

- A Herbrand structure that is not a model of Π

  $\mathcal{M}_4 = \{Arc(b,c), Path(c,c), Path(c,b)\}$

  $Path(b,b), Path(b,c)$ are missing

- There are more candidates

- For a fixed program, Herbrand structures can be compared by set inclusion: $\mathcal{M}_1 \subsetneq \mathcal{M}_3 \subsetneq \mathcal{M}_2$, $\mathcal{M}_4 \subsetneq \mathcal{M}_3$

  $\subseteq$ is a partial order in the class of Herbrand structures

- The same applies to Herbrand models, i.e. those that satisfy Π

  In our example: $\mathcal{M}_1$ is a minimal Herbrand model of Π:

  - It is a model of Π
  - There is no other Herbrand model $\mathcal{M}$ with $\mathcal{M} \subsetneq \mathcal{M}_1$

    That is, no proper subset is also a model (check this eliminating one of the atoms in $\mathcal{M}_1$ at a time)

- The minimal model will be the intended model
- Its minimality is what we need to make recursive definitions (and transitive closure) work
- No unjustified atoms in the intended model
- An extension of the CWA

Example: Datalog program

$$R(x) \leftarrow P(x)$$
$$P(a)$$
$$Q(b)$$

- Contents of P and Q are explicitly given, and the "intentional" relation R a defined view
- A Herbrand structure representing a possible state of the world:

  $$\mathcal{M}_1 = \{P(a), Q(b), R(a), R(b)\}$$

  It says that those atoms in $\mathcal{M}_1$ are true and nothing else, e.g. that $P(a)$ is true, but not $P(b)$
- This is a model of the program, because it makes all the rules above true (check this!)
- There are models and non-models: (check them!)

Models

- $\{P(a), Q(b), R(a), R(b), P(b)\}$
- $\mathcal{M}_1 = \{P(a), Q(b), R(a), R(b)\}$
- $\mathcal{M}_0 = \{P(a), Q(b), R(a)\}$
- ...

Non-models

- $\{P(a), R(a), R(b), P(b)\}$
- $\{P(a), Q(b), R(b)\}$
- $\{Q(b), R(a), R(b)\}$
- ...

- $\mathcal{M}_1$ is a model, but in it $R(b)$ is unnecessarily true, "unjustified"

  $R(a)$ is forced to be true (to satisfy the rules), but not $R(b)$

- Instead, $\mathcal{M}_0$ contains exactly what is necessary to make the program true

  It is (set-theoretically) contained in any other model of the program

  $\mathcal{M}_0$ gives the meaning (semantics) to the program

- More generally: We are comparing models (sets of ground atoms) by set inclusion, which is a partial order

  A partial order may have none, one or several minimal elements

- Theorem: A Datalog program Π has exactly one minimal Herbrand model, denoted $\underline{\mathcal{M}}(\Pi)$
- By definition, the semantics of Π is given by $\underline{\mathcal{M}}(\Pi)$
- What is true wrt. to Π is exactly what is true in $\underline{\mathcal{M}}(\Pi)$
- Interpreted as "program Π describes world $\underline{\mathcal{M}}(\Pi)$"
- How can be compute the minimal model?
- Generating the program instantiation and checking candidates not appealing

Example: Program Π

$$
\begin{aligned}
R(x) &\leftarrow P(x) \\
P(x) &\leftarrow Q(x,y) \\
Q(a,a) &\leftarrow \\
Q(a,b) &\leftarrow
\end{aligned}
$$

- The minimal model $\underline{\mathcal{M}}(\Pi)$ of the program can be obtained bottom-up, by propagating the facts through the rules, from right to left (forward-propagation), iteratively:
- First step: $Q(a,a),\ Q(a,b) \in \underline{\mathcal{M}}(\Pi)$

  Second step: $P(a) \in \underline{\mathcal{M}}(\Pi)$

  Third step: $R(a) \in \underline{\mathcal{M}}(\Pi)$
- A fix-point has been reached; nothing new is obtained

$$\underline{\mathcal{M}}(\Pi) = \{Q(a,a), Q(a,b), P(a), R(a)\}$$

- This is general, even with recursion: The minimal model of a Datalog program can be obtained as the fix-point of the bottom-up evaluation we just described

Datalog vs. Relational Algebra/Calculus

- Almost every RA operation can be expressed by means of a Datalog program
- Examples: Defined predicate Ans collects operation results

RA	Datalog

  - Selection

    $\sigma_{X=a}(R(X,Y))$ $\qquad$ $Ans(a,Y) \leftarrow R(a,Y)$

  - Intersection (conjunction)

    $R(X,Y) \cap S(X,Y)$ $\qquad$ $Ans(X,Y) \leftarrow R(X,Y), S(X,Y)$

  - Union (disjunction)

    $R(X,Y) \cup S(X,Y)$ $\qquad$ $Ans(X,Y) \leftarrow R(X,Y)$
    $Ans(X,Y) \leftarrow S(X,Y)$

    Two rules define the union, for the two cases

  - Projection (existential quantification) (c.f. page 8)

    $\Pi_X(R(X,Y))$ $\qquad$ $Ans(X) \leftarrow R(X,Y)$

- $Ans(X) \leftarrow R(X, Y)$

  Like having an existential quantifier on Y on the RHS

  This is always the case when a variable appears in the body of the rule (the RHS), but not in the head (LHS)

- Join

  $R(X, Y) \bowtie_{Y=Z} S(Z, V)$ $\quad$ $Ans(X, Y, V) \leftarrow R(X, Y), S(Y, V)$

- Difference

  $R(X, Y) \setminus S(X, Y)$ $\quad$?????

  No negation in Datalog!

- But we do have recursion in Datalog!

  And we can do things that are (provably) impossible in RA
  E.g. defining *Ancestor* as the TC of *Parent*

- The SQL standard (SQL99 or SQL3) adopted recursive views

  Idea, semantics, and query evaluation come from Datalog (coming)

- Datalog can be extended with built-in or evaluable predicates: $=, \neq, <, \ldots$

  They have a fixed and intended interpretation

  E.g. if a and b are different constants, the atom $a = b$ is evaluated as false, whereas $a = a$ is true

  With infinite extensions if underlying domain is infinite

- They can be used in conditions in the bodies, e.g.

$$SeniorParent(x) \leftarrow Parent(x, y), Age(x, z), z > 65$$

$$Ans(x, y) \leftarrow R(x, y), x = a$$

  They are evaluated as above

Example: A Datalog program defining two intentional (virtual) relations on top of extensional relational DB

$$P(x,y) \longleftarrow Q(x,y), R(x,z,v)$$
$$Q(x,y) \longleftarrow S(x,u), M(u,y)$$

S	A	B
	a	b
	a	c
	d	c

M	B	C
	a	b
	b	c
	c	e

R	A	D	E
	a	b	t
	c	c	g
	e	f	h
	c	a	s

Variables on RHS that are not on LHS are filtered out (implicitly existentially quantified)

To create extensions (tables) for intentional predicates P and Q (if we wanted; it is not always necessary): propagate data from the underlying tables to the RHSs of the rules, and finally to the LHSs of the rules

For Q: Evaluate the RHS of the rule for Q is like posing the relational algebra query $\Pi_{AC}(S \bowtie_A M)$, propagating the obtained tuples to Q's extension, obtaining $Q = \{(a,c),(a,e),(d,e)\}$

Now that we have Q, we can compute the extension of P using the first rule

The query in the body is $\Pi_{AC}(Q \bowtie_A R)$; evaluating it propagate to the left the resulting tuples: $P = \{(a,c),(a,e)\}$

And the minimal model is computed

Notice that the instantiated rule (implication):

$$Q(a,d) \longleftarrow S(a,e), M(e,d)$$

is true, because the RHS (the antecedent) is false (those tuples are not in the extensional DB)

In spite of this, we *do not* add (a,d) to Q's extension: to make an implication true we always make the RHS true first, and then the LHS, by data propagation

This a restricted and minimal way of making implications true: we make true what is forced to be true, we never insert tuples for free, there must be a justification (the truth of the antecedent)