Data Management and Databases

Chapter 4: Recursion, Datalog, Active Rules

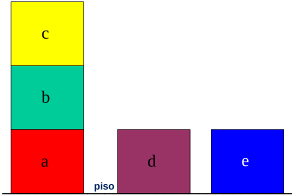
Leopoldo Bertossi

Universidad San Sebastián

Facultad de Ingeniería y Ciencias

Recursion and Query Languages

- The "blocks world":



- We can represent this blocks world as a RDB
- We can describe and query the "blocks world" with RC
  (a fragment of Predicate Logic)
- Introduce Relational Predicates: *Block*$(\cdot)$, *On*$(\cdot,\cdot)$, *Color*$(\cdot,\cdot)$, *LeftOf*$(\cdot,\cdot)$

Block	
	a
	b
	c
	d
	e

On		
	c	*b*
	b	*a*
	a	*piso*
	d	*piso*
	e	*piso*

Color		
	a	*rojo*
	b	*verde*
	c	*amarillo*
	d	*purpura*
	e	*azul*

LeftOf		
	a	*d*
	d	*e*

(for immediate left)

base tables

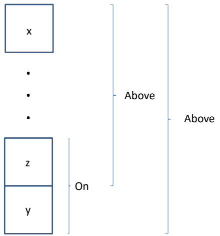
- We can use RC to define new relational predicates
  What we usually call a view definition in RDBs
- Example: Define the predicate (view) $\mathit{ClearBlock}(\cdot)$
  Intended to apply to those blocks that are clear, with nothing on top

$$\forall x(\mathit{ClearBlock}(x) \;:\longleftrightarrow\; (\mathit{Block}(x) \land \neg\exists y\, \mathit{On}(y,x))) \qquad (**)$$

  A new predicate symbol introduced in the language with its definition: the query on the RHS
- On the LHS, the newly defined predicate is introduced
  On the RHS side we have already available relations
- Now the DB can be used together with the formula in (**)
  We obtain $\{\mathit{ClearBlock}(c), \mathit{ClearBlock}(d), \mathit{ClearBlock}(e)\}$ as possibly virtual extension
- No matter what are the contents of the base tables, the view definition will always give the intended contents

- Now we want to define the predicate (view) $Above(\cdot, \cdot)$

  To be 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 view?
- A difficulty: We cannot bound a priori the number of vertical dots
- A recursive definition seems to be needed
- Notice that $Above$ is meant to be the Transitive Closure (TC) of On



- Are we not clever enough to come up with a definition in RC?
- Theorem: It is not possible to define in Predicate Logic the TC of a binary relation (the smallest transitive binary relation that includes the given one)
- Then, the same applies to RC and RA (the latter would need unbounded iteration)

Datalog

- Datalog is a logic-based language that extends (part of) RC with recursion
- As a "Datalog program", used to define queries and views on top of relational tables
- Proposed and investigated in the mid 80s
- Constructs of Datalog found their way into commercial RDBMSs (coming)
- 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
- There are newer "ontological" extensions of Datalog

- Example: The DB of the blocks world
- The view *RightOf*, for immediate right, defined by a Datalog rule:

$$RightOf(x, y) \longleftarrow LeftOf(y, x)$$

- Variables are all implicitly universally quantified ($\forall$)
- Notation from Logic Programming tradition (Prolog)
  What is being defined in on the LHS (the "head" of the rule)
- It is an implication, from right to left, but applied as- and with the semantics of a double implication
- $RightOf(e, d)$ becomes true, because the instantiated "body of the rule", $LeftOf(d, e)$, is true in the underlying DB
- A head is true only if the corresponding body is true (an "iff" then)

- Example:

Database D

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

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

- Datalog rule defining view *TopManager*, managers who make more than 4K

$$TopManager(x) \leftarrow Positions(x,z) \, , \, Salaries(x,y) \, , \, z = manager \, , \, y > 4K$$

- Here, comma stands for $\wedge$, and the variables that appear only in the body are implicitly existentially quantified

  So, think of the rule as: $TopManager(x) \leftarrow \exists z \exists y (Positions(x,z) \cdots$

- We can compute extension of the view by forward-propagating what is true on the RHS to the LHS, and nothing more:

  $TopManager[D] = \{\langle J.Page \rangle, \ \langle M.Stowe \rangle\}$ (the extension of the view on D)

- "nothing more": a form of minimization (more coming ...)

  For *V.Smith* the implication is true (body is false), but it is not collected

- It is common to list the contents of the underlying DB as a set of facts (or ground atoms) (as a part of the program)

$$
\begin{array}{rcl}
\mathit{TopManager}(x) & \longleftarrow & \mathit{Positions}(x,z), \mathit{Salaries}(x,y), \cdots \\
\mathit{Salaries}(J.Page, 5,000) & \leftarrow & \text{(usually omitted, true w/o conditions)} \\
\cdots & & \\
\mathit{Positions}(K.Stein, accountant). & &
\end{array}
$$

- The program can be seen as an extension of the RDB

- The RDB or, equivalently, the set of facts is 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

$$\underbrace{\mathit{TopManager}(x)}_{\text{what's being defined}} \leftarrow \underbrace{\mathit{Positions}(x,z), \mathit{Salaries}(x,y), z = \mathit{manager},\ y > 4K}_{\text{query to virtually extended DB}}$$

- Exercise: Actually, Datalog not needed to define the views seen so far: Define them in RC and RA and SQL

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

  This corresponds to:

Arc	V1	V2
	b	c

Path	E1	E2
	b	b
	c	c

- For most of this chapter we will represent RDBs as sets of ground atoms (tuples), as above

- Datalog program Π on top of D:

$$Path(x,z) \leftarrow Arc(x,y), \underbrace{Path(y,z)}_{\text{recursive ``call''}} \qquad (*)$$

- A recursive definition of *Path*

  Actually, a recursive extension of *Path* (it already has atoms in D)

- This is the "intended meaning" of the program

- It produces a virtual extension of D:

$$Path(b,c) \leftarrow \underbrace{Arc(b,c)}_{\surd}, \underbrace{Path(c,c)}_{\surd}$$

Path′	E1	E2
	b	b
	c	c
	b	c

- $\Pi[D] := \{Arc(b,c), Path(b,b), Path(c,c), Path(b,c)\}$ (the intended model)

- What is the semantics of a Datalog program?

  As an extension of an EDB

- What world is Π describing?

- Is there a precisely defined intended model for Π?

- We will give a "model-based" semantics to Datalog programs

  In terms of the intended models of the program

- In the case of Datalog, the potential, candidate models will be sets of ground atoms

  That is, RDBs as in the previous example

- For illustration, the semantics of Π above?

- Example: (continued) Given D and Π
- Herbrand Universe: $H := \{b, c\}$, formed by all the constants in D or Π
- Herbrand Base:

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

  Instantiate all relational predicates of D or Π on H, in all possible ways

- HB contains all the possible ground atoms that can be built with the program's language (considering D as a part of it)
- Each subset of HB is a candidate to be an intended model of $\Pi \cup D$

  2^8 subsets, each of them looking like a RDB

- What conditions should intended models satisfy?

- Given $\Pi \cup D$ and $S \subseteq HB$, what conditions should S satisfy?
  1. $D \subseteq S$: A model must extend the given EDB (the given facts)
  2. Each instantiated rule of Π must be true in S (as a usual implication)

     When the instantiated body becomes true in S, the head must be true in S

     Equivalently, when all the atoms in the body belong to S, the atom in the head must be in S as well
  3. Maybe more?
- By definition, a model satisfies conditions 1. and 2. above
- Let us check possible candidates ...

- $S_1 = \{Arc(b,c), Path(c,c), Path(c,b)\}$

  It does not satisfy 1.: $D \not\subseteq S_1$ discarded!

- $S_2 = \{Arc(b,c), Path(b,b), Path(c,c), Path(c,b)\}$

  $D \subseteq S_2$, but 2. not satisfied:

Instantiated rule: $\underbrace{Path(b,c)}_{\times} \leftarrow \underbrace{Arc(b,c)}_{\checkmark}, \underbrace{Path(c,c)}_{\checkmark}$ discarded!

- $S_3 = \{Arc(b,c), Path(b,b), Path(c,c), \underline{Path(c,b)}, Path(b,c)\}$

  $D \subseteq S_3$, what about 2.?

  Instantiated rule: $\underbrace{Path(b,c)}_{\surd} \leftarrow \underbrace{Arc(b,c)}_{\surd}, \underbrace{Path(c,c)}_{\surd}$

  What about the "unjustified" presence of $Path(c,b)$?

  It does not violate any implication, but is not implied by any

  Actually, all instantiated rules are true in S_3

  S_3 is a model! not discarded! (yet)

- Notice: $\underbrace{Path(c,b)}_{\surd} \leftarrow \underbrace{\underbrace{Arc(c,b)}_{\times}, \underbrace{Path(b,b)}_{\surd}}_{\times}$

  The implication is still true: it has a false body

- $S_4 = \{Arc(b,c), Path(b,b), Path(c,c), Path(b,c)\}$

(same as on page 9)

$D \subseteq S_4$, what about 2.?

Instantiated rule: $\underbrace{Path(b,c)}_{\surd} \leftarrow \underbrace{Arc(b,c)}_{\surd}, \underbrace{Path(c,c)}_{\surd}$

Easy to check that S_4 is a model!

- Which of S_3, S_4 is better (or preferred)?

- Given $\Pi \cup D$ and $S \subseteq HB$, what conditions should satisfy to be a preferred (or intended) model?
  1. As before
  2. As before
  3. S must be minimal model, i.e. a model, and no proper subset can be a model
- So, an intended model must:
  - Contain D
  - Make true all the possible instantiated rules:

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

  - Be minimal
- The program has many other models (find/check them!)
- How many of them are minimal?

- Theorem: A Datalog program $\Pi \cup D$ has exactly one minimal model
  Denoted: $\underline{M}(\Pi \cup D)$
- By definition, the semantics of $\Pi \cup D$ is given by $\underline{M}(\Pi \cup D)$
  What is true w.r.t. to $\Pi \cup D$ is exactly what is true in $\underline{M}(\Pi)$
- We can say that the program $\Pi \cup D$ describes the world $\underline{M}(\Pi \cup D)$
- Minimality is what will make recursive definitions (and TC) work!
- Minimality is consistent with remark at the bottom of page 7
- In the previous example, $\Pi(D)$ turns out to be the single minimal model
- How can we compute the minimal model?

- Exercise: Datalog program Π with $D = \{P(a), Q(b)\}$

$$R(x) \leftarrow P(x)$$

- Verify that $M_1 = \{P(a), Q(b), R(a), R(b)\}$ is a model
  And also that it is not minimal

- Verify that the HB itself is (always) a model

- Verify the following:

Models: (among others)	Non-Models: (idem)
$\{P(a), Q(b), R(a), R(b), P(b)\}$	$\{P(a), R(a), R(b), P(b)\}$
$\{P(a), Q(b), R(a), R(b)\}$	$\{P(a), Q(b), R(b)\}$
$M_0 = \{P(a), Q(b), R(a)\}$	$\{Q(b), R(a), R(b)\}$

- Check: M_1 is a model, but $R(b)$ is "unjustified"
- M_0 is the minimal model

- Finding the minimal model by comparison w.r.t. set inclusion with other models is not efficient

  Number of potential models in exponential is the size of HB

- There is a better, actually efficient alternative

- Theorem: Given $\Pi \cup D$, the minimal model $\underline{M}(\Pi \cup D)$ can be iteratively computed by bottom-up forward-propagation from the underlying EDB D

- We illustrate the algorithm by means of examples

- Example: Datalog program Π with EDB D

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

- That two rules share variables does not matter (they are implicitly quantified)

  We could replace the second one by
  $P(z) \leftarrow Q(z,y)$

- Computation of $\underline{M}(\Pi \cup D)$:

  Propagate the facts through the rules, from right to left (forward-propagation), iteratively:

  1. $Q(a,a),\ Q(a,b) \ \in \ \underline{M}(\Pi \cup D)$
  2. $P(a) \ \in \ \underline{M}(\Pi \cup D)$
  3. $R(a) \ \in \ \underline{M}(\Pi \cup D)$

  A fix-point has been reached; nothing new is obtained

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

- 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

- Example: Datalog program Π defining two intentional (virtual) relations on top of an EDB D

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

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
	e	f	h
	c	a	s

- Create extensions for predicates P and Q (if wanted):
  - Propagate data from EDB to the RHSs of the rules
  - Next, to the LHSs of the rules

- Evaluate RHS of Q's rule posing RA query: $\Pi_{AC}(S \bowtie_B M)$

  Propagate tuples to Q's extension: $Q = \{(a,c),(a,e),(d,e)\}$

- Compute P's extension with body query: $\Pi_{AC}(Q \bowtie_A R)$

  $P = \{(a,c),(a,e)\}$

- A minimal way of making implications true

  Making true what is forced to be true

  Inserting tuples with a justification (the truth of a body)

- We can also pose a query to the extended DB:

$$\begin{array}{rcl} Ans(x) & \longleftarrow & P(x,y), Q(x,z) \\ P(x,y) & \longleftarrow & Q(x,y), R(x,z,v) \\ Q(x,y) & \longleftarrow & S(x,u), M(u,y) \end{array}$$

- A program extended with a query
- It is computed as before (an additional iteration step)
- We already have the extensions:

$$Q = \{(a,c),(a,e),(d,e)\} \text{ and } P = \{(a,c),(a,e)\}$$

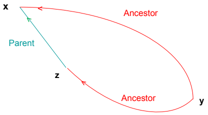
- Evaluating the first rule, we obtain the answer: $Ans = \{\langle a \rangle\}$

- Example: A Datalog program defining three intentional predicates:

$$
\begin{aligned}
\mathit{Person}(x) &\leftarrow \mathit{Parent}(x,y)\\
\mathit{Person}(y) &\leftarrow \mathit{Parent}(x,y)\\
\mathit{Grandparent}(x,z) &\leftarrow \mathit{Parent}(x,y), \mathit{Parent}(y,z)\\
\mathit{Ancestor}(y,x) &\leftarrow \mathit{Parent}(y,x) \qquad \text{(base case of recursion)}\\
\mathit{Ancestor}(y,x) &\leftarrow \mathit{Ancestor}(y,z), \mathit{Parent}(z,x)
\end{aligned}
$$

- On top of the EDB:

Parent	P	C
	juan	pablo
	adam	cain
	adam	abel
	eve	cain
	pablo	luis



- Propagate data from right to left, creating (virtual) extensions for intentional predicates

- For *Ancestor*, apply first second last rule (base case)

  Moving all the data from *Parent* into a partial extension:

  $$\mathit{Ancestor}' = \{(\mathit{juan},\mathit{pablo}),(\mathit{adam},\mathit{cain}),(\mathit{adam},\mathit{abel}),(\mathit{eve},\mathit{cain}),(\mathit{pablo},\mathit{luis})\}$$

- Now, evaluate the body of the last recursive rule, i.e. the query: $\Pi_{Anc.1,C}(\mathit{Ancestor}' \bowtie \mathit{Parent})$ (at this stage, a self-join of *Parent*)