3.3 Sets of conjunctive queries

- Let $Q_1, Q_2, \ldots, Q_n$ be conjunctive queries over a database schema $\mathcal{R}$ as follows

$$ans(\vec{U}) \leftarrow R_{i1}(\vec{U}_{i1}), \ldots, R_{in_i}(\vec{U}_{in_i}),$$

  where $n_i \geq 1$ for $1 \leq i \leq n$.
- Let $P_1, P_2, \ldots, P_m$ be conjunctive queries also over $\mathcal{R}$ as follows

$$ans(\vec{V}) \leftarrow S_{j1}(\vec{V}_{j1}), \ldots, S_{jm_j}(\vec{V}_{jm_j}),$$

  where $m_j \geq 1$ for $1 \leq j \leq m$.
- The answer-literals of Q's and P's have the same arity.

- Let $\mathcal{Q} = \{Q_1, Q_2, \dots, Q_n\}$ and $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$ be sets of conjunctive queries.
- We have $\mathcal{Q} \sqsubseteq \mathcal{P}$, $\mathcal{Q}$ is contained in $\mathcal{P}$ iff for each instance $\mathcal{I}$ of $\mathcal{R}$ there holds:

$$\bigcup_{i=1,\dots,n} Q_i(\mathcal{I}) \subseteq \bigcup_{j=1,\dots,m} P_j(\mathcal{I}).$$

- Whenever $\mathcal{Q} \sqsubseteq \mathcal{P}$ and $\mathcal{P} \sqsubseteq \mathcal{Q}$, then $\mathcal{P}$ and $\mathcal{Q}$ are *equivalent*, $\mathcal{P} \equiv \mathcal{Q}$.

Example:

$$\begin{array}{ll} Q_1 : & ans(X, Y) :- E(X, X), E(X, Y) \\ Q_2 : & ans(X, Y) :- E(X, W), E(W, Y) \\ Q_3 : & ans(X, Y) :- E(X, Y), E(X, U), E(U, Y) \end{array}$$

We have $Q_1 \sqsubseteq Q_3 \sqsubseteq Q_2$.
Moreover $\{Q_1, Q_2, Q_3\} \equiv \{Q_2, Q_3\} \equiv \{Q_2\}$.

Containment

A set $\mathcal{Q}$ is contained in a set $\mathcal{P}$ of queries, if any query in $\mathcal{Q}$ is contained in at least one query of $\mathcal{P}$.

Is this condition necessary as well?

A *containment relation* is given as mapping Ω from $\mathcal{Q}$ to $\mathcal{P}$ as follows:

$$\text{if } \Omega(Q_i) = P_j, \text{ then } Q_i \sqsubseteq P_j, \text{ where } 1 \leq i \leq n,\ 1 \leq j \leq m.$$

Theorem

Let $\mathcal{Q}$, $\mathcal{P}$ be sets of conjunctive queries. $\mathcal{Q} \sqsubseteq \mathcal{P}$ iff there exists a containment relation Ω from $\mathcal{Q}$ to $\mathcal{P}$.

Proof

(1) A containment relation Ω exists. Then $\mathcal{Q} \sqsubseteq \mathcal{P}$.
(2) $\mathcal{Q} \sqsubseteq \mathcal{P}$.
Assume there exists a Q_i such that for all P_j there holds $Q_i \not\sqsubseteq P_j$. Construct a canonical instance $\mathcal{I}_{Q_i}$ and let τ the corresponding canonical substitution.
As $\mathcal{Q} \sqsubseteq \mathcal{P}$, it follows

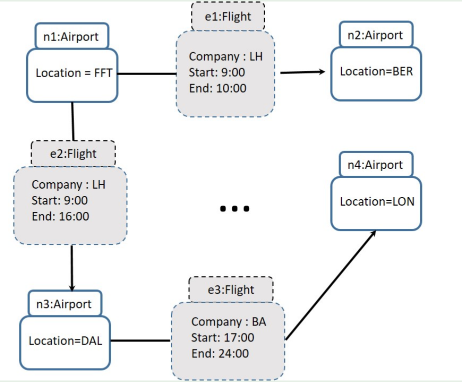
$$\tau(ans(\vec{U})) \in \bigcup_{j=1,\ldots,m} P_j(\mathcal{I}_{Q_i}).$$

Therefore there exists $j', 1 \leq j' \leq m$, such that $\tau(ans(\vec{U})) \in P_{j'}(\mathcal{I}_{Q_i})$. However then $Q_i \sqsubseteq P_{j'}$, a contradiction.

□

3.4 Datalog

Snippet of a graph DB;
Query: MATCH (X:Airport Location:"FFT") -[:Flight*]-> (Y:Airport) RETURN Y



How can we compute the answers for arbitrary instances?

Datalog: **Data**bases in **log**ic.

- Queries are expressed as rules (akin to triggers in SQL).
- Fully logic-based query language - different to SQL - resembling the *Prolog* programming language; recurrently studied in database research.

Logik

Datalog Queries: Rules

- Queries are expressed as rules.
- A rule is an implication of the form

$$H(\vec{U}) \leftarrow L_1, \ldots, L_n,$$

such that $(1 \leq i \leq n)$:

Relation

  - $L_1, \ldots, L_n$ are literals and $H(\vec{U})$ is an atom, where $H \in \mathcal{R}$.
  - The L_i's are atoms $R_i(\vec{U}_i)$ or negated atoms $\neg R_i(\vec{U}_i)$, where $R_i \in \mathcal{R}$ and
  - $\vec{U}, \vec{U}_i$ vectors of variables and constants.
  - Left to $\leftarrow$ is the *head* of the query, and to the right there is the *body*. The literals in the body are also called *subgoals*.
- A set of rules is called *program*.

Possible relational representation of a flight Graph DB instance

Relational schema: *Flight*[*Company, From, To, Start, End*];
Instance (specifying daily flight connections):

Company	*From*	*To*	*Start*	*End*
LH	FFT	BER	9:00	10:00
AA	ST	NY	9:30	16:00
LH	MUE	ROMA	10:00	12:00
BA	DAL	LON	17:00	24:00
LH	FFT	DAL	8:00	16:00
BA	LON	NY	10:00	15:00

Which destinations are reachable from Frankfurt (FFT) with direct flight connection?

FDest(X) ← Flight(_ , 'FFT', X, _ , _)

Note:

The placeholder "_" is a shortcut for a unique (undistinguished) variable that appears only in the body of the rule.

Which destinations can be reached from Frankfurt (FFT) when starting at 9:00 and changing the plane at most once?

FDest9am(X) ← Flight(_ , 'FFT', X, '9:00' , _)
FDest9am(Y) ← Flight(_, 'FFT', X, '9:00', _), Flight(_ , X, Y, _, _)

Which destinations can be reached from Frankfurt (FFT)? Recursion!

DestRec(X) ← Flight(_, 'FFT', X, _ , _)
DestRec(Y) ← DestRec(X), Flight(_, X, Y, _, _)

Which destinations can be reached from Frankfurt (FFT) taking only Lufthansa (LH) flights?

LHDestRec(X) ← Flight('LH', 'FFT', X, _ , _)
LHDestRec(Y) ← LHDestRec(X), Flight('LH', X, Y, _, _)

All destinations that can be reached from Frankfurt except those for which a Lufthansa-only (!) connection exists. Negation!

Destination(X) ← DestRec(X), ¬ LHDestRect(X)

Formal Framework

Additional Definitions

Consider a Datalog rule of the form

$$H \leftarrow L_1, \dots, L_n$$

- If $n = 0$, then its body is empty and the rule is called *fact*
- Relation symbols that appear solely on the body of rules are called *extensional*; the remaining relational symbols are called *intensional*
- Accordingly, we distinguish between the extensional database (EDB) and the intensional database (IDB)

From Datalog Rules to Datalog Programs

- A set of rules ρ is called *program* Π.
- Let Π be a program. The *dependency graph* of Π is a directed, labeled digraph (V, E) containing two types of edges (positive and negative edges) defined as follows:
  - V is the set of relational symbols appearing in the rules of ρ
  - Let P be a relational symbol of a positive literal appearing in the body of some rule ρ in Π and let Q be the relational symbol of ρ's head. Then the (positive) edge $P \longrightarrow Q$ is contained in E.
  - Let P be the relational symbol of a negative literal appearing in the body of some rule ρ in Π and let Q be the relational symbol of ρ's head. Then the (negative) edge $P \xrightarrow{\neg} Q$ is contained in E.
- We call a program *recursive*, if its dependency graph has a cycle.
- Note: the definition of recursiveness ignores edge labels; we will come back to these labels at a later point

Example: Dependency Graph

Consider the Datalog program Π defined by the rules

$$\text{DestRec}(X) \leftarrow \text{Flight}(\_, \text{`FFT'}, X, \_, \_)$$
$$\text{DestRec}(Y) \leftarrow \text{DestRec}(X), \text{Flight}(\_, X, Y, \_, \_)$$
$$\text{NotDestRec}(X) \leftarrow \text{City}(X), \neg\, \text{DestRec}(X)$$

Then the dependency graph of Π is defined as $G := (V, E)$, where

$$\begin{aligned} V &:= \{\mathrm{DestRec}, \mathrm{Flight}, \mathrm{City}, \mathrm{NotDestRec}\}, \\ E &:= \{\mathrm{Flight} \longrightarrow \mathrm{DestRec}, \mathrm{DestRec} \longrightarrow \mathrm{DestRec}, \\ &\qquad \mathrm{City} \longrightarrow \mathrm{NotDestRec}, \mathrm{DestRec} \xrightarrow{\neg} \mathrm{NotDestRec}\}. \end{aligned}$$

This program is recursive (cycle: $\mathrm{DestRec} \longrightarrow \mathrm{DestRec}$).

Definition: Active Domain

- The active domain of an instance $\mathcal{I}$, $adom(\mathcal{I})$, is defined as the set of all constants appearing in $\mathcal{I}$.
- The active domain of a Datalog program Π w.r.t. input $\mathcal{I}$, $adom(\Pi, \mathcal{I})$, is the set of all constants appearing in Π and $\mathcal{I}$.

Safe Datalog

- Let Π be a Datalog program and let $\mathcal{I}_E$ be an instance of extensional relational symbols (input) and $\mathcal{I}_A$ be an instance of the intensional relational symbol (output, i.e. set of answers).
- A rule is called *safe* if every variable appears in a *positive* literal in its body

Lemma

Let Π be a Datalog program. If every rule of Π is safe and $\mathcal{I}_E$ is finite, then the output $\mathcal{I}_A$ of Π is finite.

Example: Safe Datalog Program

DestRec(X) $\leftarrow$ Flight(_, 'FFT', X, _ , _)
DestRec(Y) $\leftarrow$ DestRec(X), Flight(_, X, Y, _, _)

clear to above and no positive literals in the query.

Example: Non-safe Datalog Program

Goal(X) $\leftarrow$ Flight(_, 'FFT', Y, _ , _)

Return, because x is not making any bound
Syntax is correct but has no sens

Example: Non-safe Datalog Program

Goal(X) $\leftarrow \neg$ Flight(_, 'FFT', X, _ , _)

Note

- The result of non-safe Datalog programs may depend on the underlying domain - we are interested to evaluate them w.r.t. the active domain, only.

Outlook: Datalog

In the following, we study (some of the) different fragments of Datalog:

Name	**Informal Description**
Datalog^{+}	Positive Datalog, i.e. Datalog without negated literals
Datalog$^{\neg}$	Datalog with positive and negated subgoals
Datalog$^{\neg\neg}$	An extension of Datalog$^{\neg}$, where we also allow head predicates to be negated
Stratified Datalog$^{\neg}$	An important subclass of Datalog$^{\neg}$, obtained from a syntactic restriction (will be defined later)
NR-Datalog$^{\neg}$	Non-recursive Datalog with negation
Datalog^{wff}, Datalog^{stable}	Datalog$^{\neg}$ beyond stratification

Datalog$^+$

Definition

A set of safe positive rules is called Datalog$^+$ program.
Datalog$^+$ rules are of the following form:

$$H(\vec{U}) \leftarrow R_1(\vec{U_1}), \ldots, R_n(\vec{U_n}),$$

where H and the R_i's are relational symbols out of $\mathcal{R}$.

Naive Evaluation Algorithm for Datalog$^+$ Programs

Goal: compute the output $\mathcal{I}_A$ of some Datalog$^+$ program Π w.r.t. the input $\mathcal{I}_E$

(1) Begin: initialize the relations of the intensional relational symbols R with $\emptyset$, i.e. $\mathcal{I}_A^0(R) = \emptyset$. Set $j := 0$.

(2) (a) Set $j := j + 1$.
Let ρ be a rule from Π of the form $H \leftarrow G$, where $H = R(a_1, ..., a_k)$ with variables or constants a_i $(1 \leq i \leq k)$. Set

$$\mathcal{I}_\rho(R) := \{(\nu(a_1), ..., \nu(a_k)) \mid (\mathcal{I}_E \cup \mathcal{I}_A^{j-1}) \models_\nu G, \nu \text{ is a variable assignment for } G\ \}$$

(b) Let R be an intensional relational symbol and let $\rho_1^R, ..., \rho_l^R$ be the rules having predicate R in their head. Put

$$\mathcal{I}_A^j(R) := \cup_{i=1}^{l} \mathcal{I}_{\rho_i}(R)$$

(2) Repeat step (2) until $\mathcal{I}_A^j(R) = \mathcal{I}_A^{j-1}(R)$ for all intensional relational symbols R.

Examples: Naive Evaluation Algorithm

- $Dest(\text{'FFT'}, X) \leftarrow Flight(\_ , \text{'FFT'}, X, \_ , \_)$

j	$\mathcal{I}_A$ (Dest)
0	$\emptyset$
1	{(FFT, BER), (FFT, DAL)}
2	{(FFT, BER), (FFT, DAL)}

- $DestRec(X) \leftarrow Flight(\_ , \text{'FFT'}, X, \_ , \_)$
  $DestRec(Y) \leftarrow DestRec(X), Flight(\_ , X, Y, \_ , \_)$

j	$\mathcal{I}_A$ (DestRec)
0	$\emptyset$
1	{BER, DAL}
2	{BER, DAL, LON}
3	{BER, DAL, LON, NY}
4	{BER, DAL, LON, NY}

Proposition

Given a Datalog$^+$ program Π, the naive evaluation algorithm always terminates.

Proof Sketch

Follows from the observation that the computation in steps (2a) and (2b) is monotonic and the finiteness of the output (Datalog$^+$ is safe).

Example *ancestor*

(1) $a(X,Y) \leftarrow p(X,Y)$
$\quad a(X,Y) \leftarrow a(X,Z), p(Z,Y)$

(2) $a(X,Y) \leftarrow p(X,Y)$
$\quad a(X,Y) \leftarrow p(X,Z), a(Z,Y)$

(3) $a(X,Y) \leftarrow p(X,Y)$
$\quad a(X,Y) \leftarrow a(X,Z), a(Z,Y)$

$p =$

parent	*kid*
Abraham	*Isaac*
Abraham	*Ishmael*
Isaac	*Jacob*
Ishmael	*Nebaioth*
Jacob	*Joseph*
Joseph	*Ephraim*

(1)

j	$\mathcal{I}_A(a)$
0	$\emptyset$

(2)

j	$\mathcal{I}_A(a)$
0	$\emptyset$

(3)

j	$\mathcal{I}_A(a)$
0	$\emptyset$

Example *same generation*

SG:

$$sg(X,X) \leftarrow p(X,Y)$$
$$sg(X,Y) \leftarrow p(X_1,X), sg(X_1,Y_1), p(Y_1,Y)$$

$p =$

parent	*kid*
Abraham	*Isaac*
Abraham	*Ishmael*
Isaac	*Jacob*
Ishmael	*Nebaioth*
Jacob	*Joseph*
Joseph	*Ephraim*

j	$\mathcal{I}_A(sg)$
0	$\emptyset$

Semi-naive Evaluation

Naive bottom-up evaluation is highly inefficient because of redundant computations.

Observation:

To derive in round $i+1$ a new, previously not derived fact, we have to use a fact having been derived in round i as a new fact.

consider a non-recursive, however infinite, version of the *ancestor* Datalog program.

$$
\begin{array}{ll}
anc': & \Delta_a^1(X,Y) \leftarrow p(X,Y) \\
 & \Delta_a^2(X,Y) \leftarrow \Delta_a^1(X,Z), p(Z,Y) \\
 & \vdots \\
 & \Delta_a^{i+1}(X,Y) \leftarrow \Delta_a^i(X,Z), p(Z,Y) \\
 & \vdots
\end{array}
$$

The final result is given by computing the union of all Δ^i, $i \geq 0$.

...this is the underlying idea of semi-naive computation of Datalog programs.