Database Theory

Filip Murlak*

March 6, 2012

Unofficial notes for the course *Database Theory*, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, 2009/2010, summer term.

Most of the material covered here comes from the book *Foundations of Databases* by S. Abiteboul, R. Hull, and V. Vianu. The book is much more fun.

You have been warned.

1 Wprowadzenie

Class 1

Bazy danych a logika

- *Schemat bazy danych* to ustalona sygnatura

$$\Sigma = \{R_1, R_2, \ldots, R_n, f_1, f_2, \ldots, f_m\}.$$

  R_i to symbole relacyjne, f_i to symbole funkcyjne. Symbolem $\mathrm{ar}(R)$ oznaczamy arność symbolu R, tzn. oczekiwaną liczbę argumentów.

- *Baza danych* to skończona struktura

$$\mathbb{A} = \langle U, R_1^{\mathbb{A}}, R_2^{\mathbb{A}}, \ldots, R_n^{\mathbb{A}}, f_1^{\mathbb{A}}, f_2^{\mathbb{A}}, \ldots, f_m^{\mathbb{A}} \rangle$$

  nad sygnaturą Σ. Prawie zawsze będziemy pomijali „wykładniki” $\mathbb{A}$, jeśli tylko będą wynikały z kontekstu.

- *Jezyk zapytań* to logika, np. rachunek predykatów, logika pierwszego rzędu (FO), FO+domknięcie przechodnie, FO+punkty stałe, MSO, itp.

- *Model danych* to podklasa struktur wraz z podstawową wiedzą o sposobie ewaluacji zapytań.

*Dziękuję Panu Mateuszowi Łupińskiemu za uważne przeczytanie notatek i wyłapanie błędów.

Model relacyjny

Rozważamy dowolne struktury relacyjne (brak symboli funkcyjnych). Schemat bazy danych, zwykle oznaczany $\mathcal{R}$, to ustalona sygnatura relacyjna. Ewaluacja zapytań przebiega przez manipulacje zbiorami krotek, czyli relacjami.

Przykładowe zapytanie:

$$Q(x,y) \leftarrow R(x,z), S(z), R(z,y)\,.$$

Model grafowy

Rozważamy wyłącznie sygnatury zawierające tylko dwuargumentowe symbole relacyjne. Strukturę nad taką sygnatura naturalnie mozna zinterpretowac jako graf skierowany z etykietowanymi krawędziami. Ewaluacja następuje poprzez eksplorację grafu. Zastosowania: *social networks*, dane biologiczne. Jest to abstrakcyjny model podobny do obiektowego i sieciowego.

Przykłady zapytań:

$$Q(x,y) \leftarrow x \xrightarrow{a^*} y, y \xrightarrow{(a+b)^*} x$$

$$Q(\pi) \leftarrow x \xrightarrow{\pi} y, \pi \in (a+b)^*$$

Model grafowy jest przykładem modelu *nawigacyjnego*, podobnie jak model drzewiasty opisany poniżej. Nawigacyjność modelu polega na tym, że podczas ewaluacji zapytań wykorzystujemy referencje do sąsiednich obiektów. Żeby zobaczyć sąsiadów węzła, nie trzeba oglądać wszystkich wierzchołków w grafie. W modelu relacyjnym nie ma referencji: konieczne jest obejrzenie całej tabeli (ang. *full scan*). Dlatego ważną częścią modelu danych jest podstawowa wiedza o sposobie ewaluacji zapytań!

Dygresja. Reakcją relacyjnych baz danych na referencje są indeksy. Taki indeks to po prostu implementacja słownika: zbalansowane drzewo, B-drzewo, itp. Czyli tak na prawdę wyszukiwanie jest logarytmiczne, a nie liniowe. Zauważmy rownież, że wskaźniki też nie daja dostepu w stałym czasie: wskaźnik to adres pozycji, którego rozmiar jest logarytmiczny.

Model drzewiasty/hierarchiczny

Dwa rodzaje obiektow (ang. *sort*), czyli dwa rozłączne uniwersa, wierzchołki i wartości. Na wierzchołkach jest jedna ustalona dwuargumentowa relacja: dziecko (rozważamy wyłącznie modele, które zinterpretowane jako graf tworzą drzewo). Ponadto mogą być relacje unarne na wierzchołkach. Na wartościach też jest jedna relacja: równość. Dodatkowo jest funkcja zwracająca wartość przechowywaną w wierzchołku: $\rho(v) = a$. Podobnie jak w przypadku modelu grafowego, obliczanie zapytań odbywa się poprzez eksplorację drzewa. Model blisko zwiazany z formatem XML.

Przykładowe zapytania (symbol $/\!/$ oznacza potomka):

$$Q(x)\colon\ a[b(x), /\!/c/b(x)]\,,$$

$$Q(x,y)\colon\ a/\!/\_\,(x)/\!/\_\,(y), x \neq y\,.$$

Typowe pytania

- Jaka jest siła wyrazu jezyka zapytań L?
- Jaka jest złożoność obliczeniowa zapytań w języku zapytań L?
- Jak szybko obliczyć zapytanie Q?

2 Zapytania koniunkcyjne

Koniunkcyjny Datalog, czyli jak o tym myśleć

Składnia. Ustalmy przeliczalny zbiór stałych $\mathcal{D}$, zwany dziedziną, oraz schemat $\mathcal{R}$. Formalnie, bedziemy pracowali z sygnaturą $\Sigma = \mathcal{R} \cup \mathcal{D}$. Zapytaniem w koniunkcyjnym Datalogu jest

$$Q(\bar{u}) \;\leftarrow\; R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)\,, \bar{v} = \bar{w} \tag{1}$$

gdzie

- $n \geq 0$, $R_i \in \mathcal{R}$, $Q \notin \mathcal{R}$,
- $\bar{u}, \bar{u}_1, \ldots, \bar{u}_n, \bar{v}, \bar{w}$ to krotki zmiennych i stałych z $\mathcal{D}$,
- $|\bar{u}_i| = \mathrm{ar}(R_i)$, $|\bar{v}| = |\bar{w}|$.

Semantyka. Niech $\mathbb{A}$ bedzie bazą danych o schemacie $\mathcal{R}$, używającą wyłącznie stałych z $\mathcal{D}$, tzn. $U \subseteq \mathcal{D}$. Definiujemy

$$Q^{\mathbb{A}} = \{\theta(\bar{u}) \;\big|\; \theta \colon \mathrm{var}(Q) \to \mathcal{D},\;\; \forall i\; \theta(\bar{u}_i) \in R_i^{\mathbb{A}}\,, \theta(\bar{v}) = \theta(\bar{w})\}$$

Bezpieczne zapytania. Zwroćmy uwagę, że odpowiedź zwrócona przez zapytanie może być nieskończona. Tymczasem chielibyśmy rozważać wyłącznie skończone bazy danych. Powodem nieskończoności sa "nieograniczone" zmienne. Wprowadzimy dodatkowy warunek bezpieczeństwa: każda zmienna w $\bar{u}$ musi być połączona równościami z jakąś zmienną w $\bar{u}_i$ lub z jakąś stałą z $\mathcal{D}$ (lub sama występować w $\bar{u}_i$).

Na oznaczenie powyższego języka zapytań będziemy używali $\mathrm{CQ}^{=}$.

Rachunek koniunkcyjny, czyli co to znaczy

W tej chwili jest to tylko inna składnia dla $\mathrm{CQ}^{=}$, ale potem różnice będą większe. Zapytania w rachunku koniukcyjnym są postaci

$$\{u \;\big|\; \varphi\}$$

gdzie φ to formuła FO nad $\mathcal{R}$ i $\mathcal{D}$ używająca wyłącznie $\exists$, $\wedge$, taka że $\mathrm{var}(u) = \mathrm{fvar}(\varphi)$. Podobnie jak w przypadku koniunkcyjnego Datalogu, musimy założyć, że zmienne są używane w bezpieczny sposób.

Zapytanie (1) można wyrazić w rachunku koniunkcyjnym jako

$$\{\bar{u} \mid \exists x_1 \ldots \exists x_n R_1(u_1) \wedge \cdots \wedge R_n(u_n) \wedge \bar{v} = \bar{u}\}\,.$$

Aby wyrazić zapytanie w rachunku koniunkcyjnym za pomocą Datalogu należy najpierw wyciągnąć kwantyfikatory na zewnątrz, a potem już jest bardzo łatwo.

Algebra SPC, czyli jak to obliczyć

Mamy do dyspozycji trzy operacje:

- selekcja
$$\sigma_{j=a}A = \{u \in A \mid u_j = a\}\,,$$
$$\sigma_{j=k}A = \{u \in A \mid u_j = u_k\}\,;$$
- rzut (implementuje też permutacje i powtórki)
$$\pi_{j_1,\ldots,j_k}A = \{(u_{j_1}, \ldots, u_{j_k}) \mid \bar{u} \in A\}\,;$$
- produkt kartezjański
$$A \times B = \{(u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_m) \mid \bar{u} \in A\,,\ \bar{v} \in B\}\,.$$

Składnia. Wyrażenia SPC nad schematem $\mathcal{R}$ to najmniejsza rodzina zawierająca symbole relacyjne $\mathcal{R}$ i wyrażenia postaci $\{(a)\}$ dla $a \in \mathcal{D}$ oraz zamknięta na operacje $\sigma, \pi, \times$.

Semantyka. Aby obliczyć wartość wyrażenia SPC w strukturze $\mathbb{A}$, interpretujemy symbole z $\mathcal{R}$ jako relacje w strukturze, a potem postępujemy zgodnie z definicjami operacji.

Rownoważność trzech formalizmów

Lemma 1. $\mathrm{CQ}^{=}$ *jest zamknięty na złożenia.*

Proof. Proste ćwiczenie. □

Dwa zapytania sa *równoważne*, gdy dają ten sam wynik na wszytkich bazach danych (skończonych!). Jezyki zapytań są równoważne, gdy dla każdego zapytania w jednym jezyku istnieje rownoważne zapytanie w drugim jezyku, i *vice versa.*

Theorem 1. *Algebra SPC jest równoważna* $\mathrm{CQ}^{=}$.

Proof. ($\Rightarrow$) Indukcja po budowie wyrażenia SPC.

- $\{(a)\}$ zamieniamy na $Q(a)$.

- $P \times R$ zamieniamy na

$$Q(\bar{x}, \bar{y}) \leftarrow Q_P(\bar{x}), Q_R(\bar{y}),$$

Q_P, Q_R mamy z założenia indukcyjnego. Korzystamy z zamkniętości na złożenie i gotowe.

- $\sigma_{j=a}P$ zamieniamy na

$$Q(\bar{x}) \leftarrow Q_P(\bar{x}), x_j = a,$$

Q_P otrzymujemy z założenia indukcyjnego.

- $\pi_{j_1,j_2,\ldots,j_k}P$ zamieniamy na

$$Q(x_{j_1}, x_{j_2}, \ldots, x_{j_k}) \leftarrow Q_P(\bar{x}),$$

Q_P z założenia indukcyjnego.

($\Leftarrow$) Bez zmniejszenia ogólności możemy założyć, że zapytanie w Datalogu jest postaci

$$Q(\bar{u}) \leftarrow R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n), \bar{v} = \bar{w}$$

gdzie krotki $\bar{u}_i$ zawierają tylko zmienne i każdą tylko raz. Lekko nadużywając notacji, odpowiednie wyrażenie SPC można zapisać jako

$$\pi_{\bar{u}}\sigma_{\bar{v}=\bar{w}}R_1 \times R_2 \times \cdots \times R_n,$$

gdzie selekcja narzuca równości $\bar{v} = \bar{w}$, a rzut wybiera odpowiednie kolumny. □

Ćwiczenia

1. Przykłady zapytań w różnych formalizmach. [1, Ex 4.1]
   - Kto jest reżyserem filmu *Szepty i krzyki*?
   - W których kinach wyświetlają film *Szepty i krzyki*?
   - Jaki jest adres i telefon kina *Muranów*?
   - Wypisz nazwy i adresy kin wyświetlających filmy Bergmana.
   - Czy w Warszawie gdzieś grają jakiś film Bergmana?
   - Wypisz takie pary osób, że pierwsza reżyserowała drugą i odwrotnie.
   - Wypisz reżyserów, którzy grali w swoim filmie.
   - Wypisz pary aktorów, którzy zagrali w tym samym filmie.
   - Wypisz parę ⟨'Czas apokalipsy', 'Coppola'⟩.
2. Czy z $\mathrm{CQ}^{=}$ można wyeliminować równość? (Wskazówka: Każde wyrażenie CQ jest spełnialne.) Wykaż, że $\mathrm{CQ}^{=}$ jest równoważny $\mathrm{CQ} \cup \{\emptyset\}$, gdzie $\emptyset$ to zapytanie zawsze zwracające zbiór pusty. [1, Ex 4.5]

3. Algebra SPJR jest „równoważna” algebrze SPC.

4. Przecięcie relacji da się symulować w SPC.

5. Różnica relacji nie jest wyrażalna w SPC.
   Monotoniczność.

6. Zbior operacji SPC nie jest redundantny (prawie). [1, Ex 4.29]

7. Postać normalna SPC: $\pi...\sigma...R_1 \times R_2 \times \cdots \times R_n$.

8. Suma nie wyraża się w SPC.
   Każde wyrażenie SPC albo zawsze zwraca najwyżej jedną krotkę, albo zwraca dowolnie dużo krotek. Wykorzystując postać normalną $\pi...\sigma...R_1 \times R_2 \times \cdots \times R_n$ zauważamy, że każde wyrażenie ma kolumny ustalone do co najwyżej jednej wartości i kolumny wolne, choc może połączone równością z innymi wolnymi kolumnami. (Inna własność: wartości z różnych kolumn nie spotykają się nigdy w tej samej kolumnie.)

9. Pokaż, że po dodaniu sumy do SPC oraz alternatywy do $\mathrm{CQ}^{=}$, oba formalizmy dalej są równoważne. Zastanów się, jakie trudności w zdefiniowaniu zapytań bezpiecznych niesie dodanie alternatywy do rachunku koniunkcyjnego.

3 Complexity of conjunctive queries

Class 2

Three flavours of complexity are considered in database related problems: data complexity, expression complexity, and combined complexity. In case of query evaluation they are defined as follows.

Combined complexity. Input: $\mathbb{A}$, Q, $\bar{a}$. Output: $\bar{a} \in Q^{\mathbb{A}}$?

Data complexity. Fixed $Q, \bar{a}$. Input: $\mathbb{A}$. Output: $\bar{a} \in Q^{\mathbb{A}}$?

Expression complexity. Fixed $\mathbb{A}$. Input: $Q, \bar{a}$. Output: $\bar{a} \in Q^{\mathbb{A}}$?

Combined complexity is the usual complexity of a problem. Out of the three, data complexity is especially important, as queries tend to be much smaller that databases. Note that if $\mathbb{A}, Q, \bar{a}$ are fixed, the problem can be solved in constant time.

Theorem 2. *Data complexity of conjunctive queries is in* P*, expression and combined complexity are* NP*-complete.*

Proof. Data complexity is in P*.* Check all possible valuations of variables of Q with constants used in $\mathbb{A}$. There is only polynomially many of such valuations, and each is of polynomial size. Checking if a single valuation makes the body of the query true can be done in P.

Combined complexity is in NP*.* Simply guess a valuation and check if it makes the body of the query true.

Expression complexity is NP*-hard.* We provide a LOGSPACE reduction from 3CNF-SAT. Take a 3CNF formula

$$\varphi = (X_1 \vee Y_1 \vee Z_1) \wedge (X_1 \vee Y_1 \vee Z_1) \wedge \cdots \wedge (X_n \vee Y_n \vee Z_n)\,,$$

with X_i, Y_i, Z_i either a variable from $x_1, \ldots, x_k$ or its negation. Define

$$\begin{aligned} Q \leftarrow\ & C(X_1, Y_1, Z_1), \ldots, C(X_n, Y_n, Z_n), \\ & N(x_1, \bar{x}_1), \ldots, N(x_k, \bar{x}_k)\,; \\ \mathbb{A} = \{& C(0,0,1), C(0,1,0), C(1,0,0), C(0,1,1), C(1,0,1), C(1,1,1), \\ & N(0,1), N(1,0)\}\,. \end{aligned}$$

It is not difficult to see that every substitution making Q true gives a satisfying valuation for the SAT instance φ. Indeed, every clause is mapped in such a way that one of its literals is 1. □

Remark. We can prove that query evaluation is NP-hard even for queries that do not use the same variable twice in a single atom: in the reduction above N-atoms never use the same variable twice, and if some C-atom does, replace it with $C'(U, V)$, or even $C''(U)$, and add $C'(0,1), C'(1,0), C'(1,1), C''(1)$ to $\mathbb{A}$. We will use this fact later.

Exercises

1. How would you evaluate efficiently $\pi_{i_1, i_2, \ldots, i_k} A$?
   Crux: removing duplicates. Solution: sort the set of tuples.

2. Let $A \bowtie_F B$ denote $\sigma_F A \times B$ for any conjunction of equalitites F. How can one evaluate $A \bowtie_F B$ efficiently?
   Naïve: take the product and then scan it selecting the tuples that satisfy F, $\mathcal{O}(mn)$.
   Smarter: *sort-merge* in $\mathcal{O}(\max(m \log m, n \log n, \text{size of output}))$.

4 Local optimization for CQs

Query rewriting techniques

We present a simple technique based on algebraic equivalences. Let us look at our favourite example:

$$\begin{aligned} \mathit{ans}(\mathit{cinema}, \mathit{address}) \leftarrow\ & \mathit{Movie}(\mathit{title}, \text{`Bergman'}, \mathit{actor})\,, \\ & \mathit{Screens}(\mathit{cinema}, \mathit{title}, \mathit{time})\,, \\ & \mathit{Location}(\mathit{cinema}, \mathit{address}, \mathit{telephone})\,. \end{aligned}$$

When expressed directly in SPC algebra, the query looks like this:

$$\pi_{4,8} \sigma_{2=\text{`}\mathit{Bergman}\text{'}}((\mathit{Movie} \bowtie_{1=2} \mathit{Screens}) \bowtie_{4=1} \mathit{Location})\,.$$

With the query we can asociate the *query tree*, i.e., the parse tree of the SPC expression.

The idea of this approach is to modify the query locally, using simple rules that guarantee equivalence of the obtained query. Two heuristic rules of the thumb are:

- move selects down the query tree; and
- project out the columns you do not need as early as possible, i.e., as low as possible in the query tree.

For the example above this results in the following expression:

$$\pi_{2,3}\left[(\pi_2\left[(\pi_1\sigma_{2='Bergman'}Movie)\bowtie_{1=2}\pi_{1,2}Screens\right])\bowtie_{1=1}\pi_{1,2}Location\right].$$

Full list of rewriting rules is summarised below:

$$\begin{aligned}
\sigma_F\,\sigma_G\,q &\leftrightarrow \sigma_{F\wedge G}\,q\\
\pi_{\bar{j}}\,\pi_{\bar{k}}\,q &\leftrightarrow \pi_{\bar{\ell}}\,q\\
\sigma_F\,\pi_{\bar{\ell}}\,q &\leftrightarrow \pi_{\bar{\ell}}\,\sigma_G\,q\\
q\bowtie p &\leftrightarrow p\bowtie q\\
\sigma_F(q\bowtie_G p) &\rightarrow (\sigma_F\,q)\bowtie_G p\\
\sigma_F(q\bowtie_G p) &\rightarrow q\bowtie_G(\sigma_{F'}\,p)\\
\sigma_F(q\bowtie_G p) &\rightarrow q\bowtie_{G'} p\\
\pi_{\bar{\ell}}(q\bowtie_G p) &\rightarrow (\pi_{\bar{\ell}}\,q)\bowtie_{G'} p\\
\pi_{\bar{\ell}}(q\bowtie_G p) &\rightarrow q\bowtie_{G'}(\pi_{\bar{k}}\,p)
\end{aligned}$$

Note that some rules are two-way. Usually, the system tries to propose several reasonable rewritings of the query, and then chooses the cheapest one based on some analysis of the data.

Sideways information passing

Let us concentrate on expressions of the form

$$\pi_{\bar{\ell}}\,\sigma_F(R_1\times R_2\times\cdots\times R_n)\,,$$

corresponding to select-from-where queries is SQL. A heuristic method used by the INGRES system was aimed to find an equivalent expression of the form

$$\pi_{\bar{\ell}}\,\sigma_{F'}(R_{i_1}\bowtie_{F_1} R_{i_2}\bowtie_{F_2}\ldots\bowtie_{F_{n-1}} R_{i_n})\,,$$

that gives the smallest intermediate results, assuming that joins are evaluated from left to right.

To describe the method, let us present the query as

$$ans(\bar{x})\ \leftarrow\ R_1(\bar{y}_1), R_2(\bar{y}_2),\ldots,R_n(\bar{y}_n), C_1, C_2,\ldots,C_m$$

where R_i are relational atoms, and C_i are constraint atoms (e.g., inequalitites, order constraints, or even arithmetics of some sort).

The *sip-graph* of a query puts edges between atoms if the atoms share some variable.

A *sip-strategy* is an ordering of all atoms $A_1, A_2, \ldots$ such that for each j one of the following conditions holds

- A_j uses a constant;
- A_j is a relational atom and uses a variable already used in $A_1, A_2, \ldots, A_{j-1}$;
- A_j is a constraint atom and all its variables are used in $A_1, A_2, \ldots, A_{j-1}$.

If the sip-graph of the query is not connected, then we have a set of sip-strategies.

The intuitive reason why this should improve the evaluation time is that "smallness" of intermediate results originates in constants, and propagates via common variables. Consider an example

$$ans(\bar{z}) \leftarrow P(2, v), Q(5, w, x), R(v, w, y), S(x, y, z), v \leq w$$

A sip strategy is $P(2, v), R(v, w, y), Q(5, w, x), S(x, y, z), v \leq w$. As $P(a, v)$ uses a constant, there will be few fitting tuples in the database. Since $R(v, w, y)$ shares the variable v with $P(2, v)$, the number of tuples agreeing on v with the previously selected set of tuples will be small again, and so on.

5 Global optimization for CQs

Query containment

In order to optimize queries we need to be able to decide if two queries are equivalent. In fact, it is enough to consider *query containment*:

$$Q \subseteq Q' \quad \text{iff} \quad Q^{\mathbb{A}} \subseteq Q'^{\mathbb{A}} \text{ for every } \mathbb{A}.$$

Clearly Q and Q' are equivalent iff $Q \subseteq Q'$ and $Q' \subseteq Q$.

Homomorphism Theorem

A *homomorphism* $h : Q \to Q'$ is a function $h : \operatorname{Var} Q \to \operatorname{Var} Q' \cup \mathcal{D}$ such that if $R(u)$ occurs in Q, then $R(h(u))$ occurs in Q'. The symbol $h(u)$ stands for $(h(u_1), h(u_2), \ldots, h(u_r))$ where $u = (u_1, u_2, \ldots, u_r)$ and $h(c) = c$ for all $c \in \mathcal{D}$ (For simplicity we assume that the head atoms use the same relation symbol, say $O \notin \mathcal{R}$).

Theorem 3. *For $Q, Q' \in$ CQ,*

$$Q \subseteq Q' \quad \textit{iff} \quad \textit{there exists a homomorphism } h : Q' \to Q.$$

Proof. Suppose there exists a homomorphism h. Suppose $\bar{a} \in Q^{\mathbb{A}}$. Then, for some substitution $\theta : \mathrm{Var}(Q) \to \mathcal{D}$, the head of $Q\theta$ is $O(\bar{a})$ and every atom K from the body of Q satisfies $K\theta \in \mathbb{A}$. But then, every atom K' from the body of Q' satisfies $K'(h \circ \theta) \in A$, and the head of $Q'\theta$ is $O(\bar{a})$.

Conversely, assume that $Q \subseteq Q'$. Suppose that Q is as

$$O(u) \leftarrow R_1(u_1), \ldots, R_k(u_k)$$

and let

$$\mathbb{A}_Q = \{R_1(u_1), \ldots, R_k(u_k)\}.$$

For instance,

$$O(x, y) \leftarrow R(x, z), T(z), R(z, y)$$

turns into

$$\mathbb{A}_Q = \{R(x, z), T(z), R(z, y)\}.$$

More formally, $\mathbb{A}_Q = \langle \{x, y, z\}, R, T \rangle$, where $R = \{(x, z), (z, y)\}$, $T = \{(z)\}$.

Clearly $u \in Q^{\mathbb{A}_Q}$. Hence, $u \in Q'^{\mathbb{A}_Q}$. This means that there exists a substitution θ, such that for every atom K' from the body of Q', $K'\theta \in \mathbb{A}_Q$ and $u'\theta = u$. But this is exactly the homomorphism we were looking for. □

Query minimization

A query is *minimal* iff there is no equivalent query with fewer atoms.

Corollary 1. *For every query $Q \in$ CQ, there exists an equivalent minimal query Q' that can be obtained by erasing some atoms from Q.*

Proof. Let Q'' be a minimal query equivalent to Q (it obviously exists). Then there exist homomorphisms $h : Q \to Q''$, and $g : Q'' \to Q$. Consider the image of $h \circ g$. It is a query that can be obtained from Q by removing atoms. And as it is an image of Q'', it is not larger than Q''. Hence, it is minimal. □

Corollary 2. *All minimizations of a fixed query $Q \in$ CQ are isomorphic.*

Proof. Exercise. □

Complexity of query containment

Proposition 1. *Query containment is* NP*-complete for CQs.*

Proof. We show that existence of homomorphism is NP-complete. In NP: Guess a homomorphism and verify.

Hardness: Take a database $\mathbb{A}$, and a query Q. Define $Q' \leftarrow B$, where B is a conjunction of all the tuples in $\mathbb{A}$. It is easy to see that substitutions yielding Q true in $\mathbb{A}$ are exactly homomorphisms from Q to Q'. □

So, most probably there is no polynomial algorithm for query containment. A good thing is that the algorithm is only exponential in the size of the query, and queries tend to be small. If the database is large, it pays off to minimize

Remark. By Theorem 3 and Proposition 1 the following problem is NP-complete: given two arbitrary finite structures $\mathbb{A}, \mathbb{B}$ decide if there is a homomorphism $h : \mathbb{A} \to \mathbb{B}$.

Exercises

1. Minimize the query $Q(x, 4, z) \leftarrow R(x, 4, z_1), R(x_1, 4, z_2), R(x_1, 4, z)$.
   $Q(x, 4, z) \leftarrow R(x, 4, z_1), R(x_1, 4, z)$; one join saved.

2. All minimizations are isomorphic.
   By equivalence there are homomorphisms both ways, g, h. Suppose some variable is not used in the image of h. Then tuples using this variable are not in the image either. $h \circ g$ shows that the image of h is equivalent, so it cannot be smaller. Contradiction.

3. We write $Q \subset Q'$ iff $Q \subseteq Q'$ and $Q \not\equiv Q'$. Find examples of queries $Q_1, Q_2, Q_3, \ldots$ over a single relation, without constants such that

   (a) $Q_1 \supset Q_2 \supset Q_3 \supset \ldots$,
   Q_i says there is a directed path of length i

   (b) $Q_1 \subset Q_2 \subset Q_3 \subset \ldots$,
   Q_i says there is a (directed) cycle of length 2^i.

   (c) $Q_i \not\subseteq Q_j$ and $Q_i \not\supseteq Q_j$ for $i \neq j$.
   Q_i says there is a (directed) cycle of length p_i, where p_i is the ith prime.

4. (*) Same as above but for typed queries (A query is typed if it never puts the same variable in two different positions in the relation). You may use constants if you need.
   (b) zig-zag query:

   $$Q_i(x_0, x_{2i+1}) \leftarrow E(x_0, x_1), E(x_2, x_1), E(x_2, x_3), E(x_4, x_3), \ldots, E(x_{2i}, x_{2i+1}) .$$

   (c) it is easy to find arbitrarily large finite set of incomparable queries among

   $$Q_{ij}(x, x', y, y') \leftarrow Q_i(x, x'), Q_j(y, y') .$$

5. Show that query containment is decidable for UCQs, i.e., for given CQs $Q_1, Q_2, \ldots, Q_n$ and $P_1, P_2, \ldots, P_m$, decide if

   $$Q_1 \cup Q_2 \cup \cdots \cup Q_n \subseteq P_1 \cup P_2 \cup \cdots \cup P_m .$$

   Hint: Show that $Q_i \subseteq P_1 \cup P_2 \cup \cdots \cup P_m$ iff $Q \subseteq P_j$ for some j.

6 Computing with acyclic joins

Class 3

We are using the named variant of the algebra: SPJR. We will identify the relation names with sets of their column names: $R = \{A, B, C\}$ means that R has three columns, A, B, and C. While in the unnamed variant tables viewed as sets of tuples, in the named variant they are viewed as sets of functions from R to $\mathcal{D}$ (the data domain). A database over the signature $\mathcal{R}$ is usually called an *instance* of $\mathcal{R}$. Similarly for tables and relation names.

In the named variant, the role of Cartesian product ($\times$) is played by join. For an instance I of R and an instance J of S define the join of I and J as

$$I \bowtie J = \{u \cup v \mid u \in I, v \in J,\ u \text{ and } v \text{ agree on } R \cap S\}\,.$$

We are only interested in queries of the form

$$\pi_X(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k)\,.$$

We start with the worst case analysis and then move to a tractable case.

Worst case

Proposition 2. *It is* NP*-complete to decide if* $t \in \pi_X(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k)$ *for a given tuple* t*, relations* R_i *and a set of column identifiers* X*. (Even if there is at most one tuple in the result.)*

Proof. We will reduce query evaluation. Take a database $\mathbb{A}$ and a query $Q \leftarrow R_1, \ldots, R_n$. Recall we can assume that Q never uses a variable twice in a single atom (see the remark after Theorem 2). For each variable x occurring in q, let X be a fresh attribute (column name).

For every atom $R(x_1, \ldots, x_p)$ define $A_{R(x_1,\ldots,x_p)}$ as the relation R from $\mathbb{A}$ with the attributes renamed to $X_1, \ldots, X_p$. It is easy to see that Q is satisfied in $\mathbb{A}$ iff $\pi_\emptyset(\bowtie\{I_T | T \in Q\})$ contains the empty tuple $\langle\rangle$. □

The source of hardness

Consider $\mathcal{R} = \{R_1, R_2, \ldots, R_{2n+1}\}$ with $R_i = \{A_i, A_{i+1}\}$ for $1 \leq i \leq 2n$ and $R_{2n+1} = \{A_{2n+1}, A_1\}$. Let I be an instance of $\mathcal{R}$ where each R_i is interpreted as $I_i = \{a, b\} \times \{0, 1\} \cup \{0, 1\} \times \{a, b\}$. Then $I_1 \bowtie I_2 \bowtie \ldots \bowtie I_{2n+1}$ is empty, but $I_1 \bowtie I_2 \bowtie \ldots \bowtie I_{2n}$ has exponential size. In fact, the join of any $2n$ of the tables I_i is exponential and there seems to be no way to compute the full join without going through exponential intermediate results. But some times it is possible. The crucial role is played by the order in which we perform the joins, and project out the unnecessary columns. A recipe for such an economic order will be given by so called join-trees of queries. Whenever a query has a join tree, it can be evaluated in P.

What makes some queries easy, and some difficult? We will see that the underlying property is a certain notion of acyclicity of the schema. (Clearly, there is a cycle in the schema $\mathcal{R}$ described above.)

Tearing of ears and building join trees

A *join tree* for a schema $\mathcal{R}$ is an undirected graph $T = (\mathcal{R}, E)$ such that

- T is a tree,
- the edge between R and S is labelled with the attributes from $R \cap S$,

- if $A \in R \cap S$, then each label on the (unique) path between R and S includes A.

The algorithm for computing joins will use the join tree as a scenario of subsequent joins. Let us now see how to compute a join tree.

A *hypergraph* $\mathcal{F} = (V, F)$ consists of a set of vertices V and a set of hyperedges edges $F \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$. Thus every hyperedge $f \in F$ is a non-empty set of vertices. An *ear* of $\mathcal{F}$ is a hyperedge $f \in F$ such that for some distinct $f' \in F$ no vertex of $f \setminus f'$ is in any other edge, i.e.,

$$f \cap \bigcup (F \setminus \{f\}) \subseteq f \cap f' .$$

As a special case, if f is disjoint from the rest of the graph, it is also an ear. The edge f' above is called a witness for f being an ear.

The *GYO algorithm* removes ears as long as there are any, adding edges between ears and their witnesses.

```
Let T = (⋃R, ∅), F = (⋃R, R).
While possible
   find an ear f in F = (V, F) and its witness f'
   replace F with (⋃(F \ {f}), F \ {f})
   add the edge {f, f'} to T
```

Theorem 4.

1. *If GYO fed with $\mathcal{R}$ terminates with $\mathcal{F} = (\emptyset, \emptyset)$, the returned T is a join tree for $\mathcal{R}$.*

2. *If there exists a join tree for $\mathcal{R}$, GYO fed with $\mathcal{R}$ terminates with $\mathcal{F} = (\emptyset, \emptyset)$ and a join tree T for $\mathcal{R}$.*

Proof. (1) Assume that GYO terminates with $\mathcal{F} = (\emptyset, \emptyset)$ and some graph T. Let $S_1, S_2, \ldots, S_n$ be the order in which relations are removed from $\mathcal{R}$ as ears, and let $S_{w(i)}$ be the witness for S_i. Note that it always holds that $w(i) > i$. The edges in T are $\{S_i, S_{w(i)}\}$ for $i = 1, 2, n-1$. If we direct the edges from S_i to $S_{w(i)}$ we see that the graph has no cycles: each node except S_n has a single outgoing egde, and the indices increase strictly along the edges. The graph is connected, because from every node S_i we get to S_n following the path $S_i \to S_{w(i)} \to S_{w^2(i)} \to \ldots$.

Let us now check that the third condition from the definition of join-tree is satisfied. Take S_i and S_j such that $A \in S_i \cap S_j$ for some column A. Consider paths

$$S_i \to S_{w(i)} \to S_{w^2(i)} \to \cdots \to S_{w^k(i)} = S_n ,$$
$$S_j \to S_{w(j)} \to S_{w^2(j)} \to \cdots \to S_{w^\ell(j)} = S_n .$$

Assume that $A \in S_n$. For each p, if $A \in S_{w^p(i)}$ then $A \in S_{w^{p+1}(i)}$ because, being a witness, $S_{w^{p+1}(i)}$ must contain $S_{w^p(i)} \cap S_n$. Hence, $A \in S_{w^p(i)}$ for each p. Similarly $A \in S_{w^q(j)}$ for all q. This gives a path we are looking for.

Assume that $A \notin S_n$. Let p and q be minimal such that

$$A \in S_{w^p(i)} \setminus S_{w^{p+1}(i)} \quad \text{and} \quad A \in S_{w^q(j)} \setminus S_{w^{q+1}(j)} .$$

Suppose $w^p(i) < w^q(j)$. Being a witness, $S_{w^{p+1}(i)}$ should contain $S_{w^p(i)} \cap S_{w^q(j)}$ – a contradiction. The case $w^p(i) > w^q(j)$ is symmetric. Hence, $w^p(i) = w^q(j)$ and we get the path from S_i to S_j via $S_{w^p(i)} = S_{w^q(j)}$.

(2) First, we show that if T is a join-tree for $\mathcal{R}$ then some sequence of choices in GYO gives T. Observe that each node R in T is an ear in $\mathcal{R} \setminus \{\text{relations in the subtree of } T \text{ rooted at } R\}$, and that R's parent in T is a witness. The strategy for GYO is to follow the structure of T from the leaves up.

Different choices may lead to a different join-tree, but not a failure. Indeed, suppose that at some point GYO arrives at a non-empty hypergraph $\mathcal{F}$. Let R be the first relation from $\mathcal{F}$ in the sequence of ears that leads to T. We will prove that R is an ear in $\mathcal{F}$.

Let P be the parent of R in T. As all relations in the subtree of T rooted at R have been removed, R is an ear in in $\mathcal{F} \cup \{P\}$ with a witness P. If $P \in \mathcal{F}$, we are done. If P has already been removed, we find a witness for R by applying the following claim, possibly several times:

If W is a witness for R in $\mathcal{F} \cup \{W\}$ and W has been removed from $\mathcal{F}$, witnessed by W', then W' is witness for R in $\mathcal{F} \cup \{W'\}$.

Since W and W' are witnesses for R and W, respectively, we have

$$R \cap \bigcup(\mathcal{F} \setminus \{R\}) \subseteq W \quad \text{and} \quad W \cap \bigcup \mathcal{F} \subseteq W' .$$

It follows that $R \cap \bigcup(\mathcal{F} \setminus \{R\}) \subseteq W'$, which means that W' is a witness for R in $\mathcal{F} \cup \{W'\}$. □

Removing dangling tuples

We already know that in order to evaluate the query efficiently, we need to perform the joins and projections in the right order, so that unnecessary columns are not taken along the computation. But there is also another reason why an evaluation scenario might be costly when compared with the size of the output: some tuples in the intermediate results might not contribute to the final result, because they have no matching tuples in the remaining relations (tuples match, if they agree on common attributes). We shall remove such tuples beforehand.

A database $\mathbb{A}$ over $\mathcal{R} = \{R_1, \ldots, R_n\}$ is *(globally) consistent* iff for each i

$$\pi_{R_i}(R_1^{\mathbb{A}} \bowtie \ldots \bowtie R_n^{\mathbb{A}}) = R_i^{\mathbb{A}} .$$

That is, every tuples in $R_i^{\mathbb{A}}$ has a matching tuple in $R_1^{\mathbb{A}} \bowtie \ldots \bowtie R_n^{\mathbb{A}}$.

Define *semi-join* of I, and instance of R, and J, an instance of S, as

$$I \ltimes J = \pi_R(I \bowtie J) .$$

Simply, the semi-join removes from I the tuples that have no matching tuples in J.

Lemma 2.

1. $I \bowtie J = (I \ltimes J) \bowtie J = I \bowtie (J \ltimes I) = (I \ltimes J) \bowtie (J \ltimes I)$.
2. $\pi_{\bar{A}}(I \bowtie J) = \pi_{\bar{A}}(I \rtimes J)$ *whenever* $\bar{A} \subseteq \operatorname{sort} J$.

Proof. Easy exercise. □

A *semi-join program* for $\mathcal{R}$ is a sequence of semi-join commands:

$$\begin{aligned} R_{i_1} &:= R_{i_1} \ltimes R_{j_i}; \\ R_{i_2} &:= R_{i_2} \ltimes R_{j_2}; \\ &\vdots \\ R_{i_k} &:= R_{i_k} \ltimes R_{j_k}; \end{aligned}$$

A semi-join program is a *full reducer* for $\mathcal{R}$ iff it transforms each database over $\mathcal{R}$ into a consistent one.

Running a semi-join program may change the database, but by Lemma 2 it does not influence the full join $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$. This means that a full reducer removes exactly these tuples that do not contribute to the final result. But how do we get a full reducer for a database? It turns out it can be easily extracted from the join tree.

Theorem 5. *Assume that GYO terminates successfully. Let $S_1, S_2, \ldots, S_n$ be the order in which GYO removes relations from $\mathcal{R}$. Let $S_{w(i)}$ be the witness for S_i being an ear. The following is a full reducer for $\mathcal{R}$:*

$$\begin{aligned} S_{w(1)} &:= S_{w(1)} \ltimes S_1; \\ S_{w(2)} &:= S_{w(2)} \ltimes S_2; \\ &\vdots \\ S_{w(n-1)} &:= S_{w(n-1)} \ltimes S_{n-1}; \\ S_{n-1} &:= S_{n-1} \ltimes S_{w(n-1)}; \\ &\vdots \\ S_2 &:= S_2 \ltimes S_{w(2)}; \\ S_1 &:= S_1 \ltimes S_{w(1)}. \end{aligned}$$

Proof. By the *ith block* of the semi-join program above we will understand the lines from $S_{w(i)} := S_{w(i)} \ltimes S_i$ to $S_i := S_i \ltimes S_{w(i)}$. Let $R_i^j(I_i, I_{i+1}, \ldots, I_n)$ denote the content of the "variable" S_j after running the ith block over the valuation $S_i = I_i,\ S_{i+1} = I_{i+1},\ \ldots,\ S_n = I_n$.

We prove by downward induction on i that for all $j = i, i+1, \ldots, n$

$$\pi_{\mathcal{A}_j}(I_i \bowtie I_{i+1} \bowtie \ldots \bowtie I_n) = R_i^j(I_i, I_{i+1}, \ldots, I_n)\,.$$

The induction basis is $i = n-1$. Then $w(n-1) = n$. We have

$$\pi_{\mathcal{A}_n}(I_{n-1} \bowtie I_n) = I_n \ltimes I_{n-1} = R^n_{n-1}(I_{n-1}, I_n)\,,$$
$$\pi_{\mathcal{A}_{n-1}}(I_{n-1} \bowtie I_n) = I_{n-1} \ltimes (I_n \ltimes I_{n-1}) = R^{n-1}_{n-1}(I_{n-1}, I_n).$$

For $j > i$ we have

$$\begin{aligned}\pi_{\mathcal{A}_j}(\bowtie_{k=i}^n I_k) &= \pi_{\mathcal{A}_j}(I_i \rtimes \bowtie_{k=i+1}^n I_k) = \\ &= \pi_{\mathcal{A}_j}((I_i \rtimes I_{w(i)}) \bowtie (\bowtie_{k=1}^{w(i)-1} I_k) \bowtie (\bowtie_{k=w(i)+1}^n I_k)) = \\ &= R^j_{i+1}(I_i, I_{i+1}, \ldots, I_{w(i)-1}, I_{w(i)} \ltimes I_i, I_{w(i)+1}, I_{w(i)+2}, \ldots, I_n) = \\ &= R^j_i(I_i, I_{i+1}, \ldots, I_n).\end{aligned}$$

The justifications for the equalitites are: (1) $j > i$; (2) each attribute of S_i used in $S_{i+1}, \ldots, S_n$ is also used in $S_{w(i)}$; (3) induction hypothesis; (4) the semi-join is exactly what the first line of the ith block does.

For $j = i$ we have

$$\begin{aligned}\pi_{\mathcal{A}_i}(\bowtie_{k=i}^n I_k) &= \pi_{\mathcal{A}_i}\left[(\bowtie_{k=i}^n I_k) \ltimes I_i\right] = \\ &= \pi_{\mathcal{A}_i}\left[(\bowtie_{k=i}^{w(i)-1} I_k) \bowtie (I_{w(i)} \ltimes I_i) \bowtie (\bowtie_{w(i)+1}^n I_k)\right] = \\ &= I_i \ltimes \left[(\bowtie_{k=i}^{w(i)-1} I_k) \bowtie (I_{w(i)} \ltimes I_i) \bowtie (\bowtie_{w(i)+1}^n I_k)\right] = \\ &= I_i \ltimes \pi_{\mathcal{A}_{w(i)}}\left[(\bowtie_{k=i}^{w(i)-1} I_k) \bowtie (I_{w(i)} \ltimes I_i) \bowtie (\bowtie_{w(i)+1}^n I_k)\right] = \\ &= I_i \ltimes R^{w(i)}_{i+1}(I_i, I_{i+1}, \ldots, I_{w(i)-1}, I_{w(i)} \ltimes I_i, I_{w(i)+1}, I_{w(i)+2}, \ldots, I_n) = \\ &= R^i_i(I_i, I_{i+1}, \ldots, I_n).\end{aligned}$$

The justifications for the equalitites are: (1) Lemma 2 (1); (2) each attribute of S_i used in $S_{i+1}, \ldots, S_n$ is also used in $S_{w(i)}$; (3) Lemma 2 (2); (4) each attribute of S_i used in $S_{i+1}, \ldots, S_n$ is also used in $S_{w(i)}$; (5) induction hypothesis; (6) the semi-joins are exactly what the first and the last line of the ith block do. □

Tractable joins

Corollary 3. *Given a join tree for a schema $\mathcal{R}$ one can compute $\pi_X(\bowtie \mathbb{A})$ for any instance $\mathbb{A}$ over $\mathcal{R}$.*

Proof. See the proof of Corollary 6.4.6 in [1]. □

Towards acyclicity: pairwise consistency versus global consistency

Class 4

The conection between acyclicity and successful termination of GYO algorithm is via certain consistency condition.

An instance $\{I_1, I_2, \ldots, I_n\}$ of $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ is *pairwise consistent (locally consistent)* if for each i, j each tuple in I_i has a matching tuple in I_j, i.e.,

$$\pi_{R_i}(I_i \bowtie I_j) = I_i\,.$$

Note that $\pi_{R_i}(I_i \bowtie I_j) \subseteq I_i$ always holds.

Lemma 3. *Assume that every pairwise consistent instance of a schema $\mathcal{R}$ is globally consistent.*

1. *For each set of attributes U, every pairwise consistent instance of*

$$\mathcal{R}|_U = \{R \cap U | R \in \mathcal{R}\} \setminus \{\emptyset\}$$

   is globally consistent.

2. *For all $R, S \in \mathcal{R}$ such that each attribute of R is used in S, i.e. $R \subseteq S$, each pairwise consistent instance of $\mathcal{R} \setminus \{R\}$ is globally consistent.*

Proof. (1) Take a pairwise consitent instance $\mathbb{A}$ of $\mathcal{R}|_U$. Extend it to $\mathbb{A}'$ by filling up every tuple with a fixed constant 0. Clearly, any matching tuples still match after extension. Tuples using no attributes from U match all others. Hence, $\mathbb{A}'$ is pairwise consistent. By the assumption, it is also globally consistent. To conclude, note that a restriction of a globally consistent instance is always globally consistent.

(2) Similarly, but extend the instance $\mathbb{A}$ with $\pi_R(S^{\mathbb{A}})$. □

Lemma 4. *If a schema $\mathcal{R}$ has a full reducer, then every pairwise consistent instance of $\mathcal{R}$ is globally consistent.*

Proof. Take a pairwise consistent instance of $\mathcal{R}$ apply a full reducer to it. The result is globally consistent by the definition of full reducer. But on a pairwise consistent data base each semi-join program has no effect whatsoever. Hence, the original instance is globally consistent as well. □

Connection to acyclicity

Let $\mathcal{F} = (V, F)$ be a hypergraph, and $U \subseteq V$. The *restriction* of $\mathcal{F}$ to U is the hypergraph

$$\mathcal{F}|_U = (U, \{f \cap U \mid f \in F\} \setminus \{\emptyset\}) .$$

For distinct edges f, f', the set $g = f \cap f'$ is an *articulation set* of $\mathcal{F}$ iff the number of connected components in $\mathcal{F}|_{V \setminus g}$ is greater than in $\mathcal{F}$. (This generalizes the notion of articulation set for ordinary graphs.)

A hypergraph is *reduced* if no hyperedge is a proper subset of another hyperedge.

A *cycle* is a set of vertices $U \subseteq V$ such that $\mathcal{F}|_U$ (after reduction)

- is connected,
- has more than 2 edges,
- has no articulation set.

Finally, $\mathcal{F}$ is *acyclic* if it contains no cycle.

Lemma 5.

1. *For any set of vertices U, $\mathcal{F}$ is acyclic iff $\mathcal{F}|_U$ is acyclic.*
2. *A graph is acyclic iff it is acyclic after reducing.*

Proof. Almost trivial exercise. □

Theorem 6. *For every schema $\mathcal{R}$ the following conditions are equivalent:*

1. *GYO terminates successfully on input $\mathcal{R}$.*
2. *There exists a full reducer for $\mathcal{R}$.*
3. *Every pairwise consistent instance over $\mathcal{R}$ is globally consistent.*
4. *$\mathcal{R}$ is acyclic.*

Implications (1) ⇒ (2) ⇒ (3) follow by Theorem 5 and Lemma 4, respectively.

Proof of (3) ⇒ (4). By contradiction, assume that there is a schema that satisfies (3) but is not acyclic. Consider the ones with fewest relations, and out of these choose the one with fewest attributes, say $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$. We leave the following consequences of minimality as exercises:

(i) $\mathcal{R}$ is reduced,

(ii) each attribute in $\mathcal{R}$ is used in at least two relations,

(iii) $\mathcal{R}$ has no articulation set.

Let

$$\mathcal{R}' = \{R_2 \setminus R_1, R_3 \setminus R_1, \ldots, R_n \setminus R_1\}.$$

There are two main cases, depending on whether $\mathcal{R}'$ is connected or not.

$\mathcal{R}'$ is connected. In this case we construct a locally consistent instance of $\mathcal{R}$ that is not globally consistent, thus showing that $\mathcal{R}$ does not satisfy (3). Let $R_1 = \{A_1, A_2, \ldots, A_p\}$ and let $B_1, B_2, \ldots, B_q$ be the remaining attributes used in $\mathcal{R}$. Let us consider a database $\mathbb{A} = \{I_1, I_2, \ldots I_n\}$ over $\mathcal{R}$ where

$$I_i = \begin{cases} \pi_{R_i} I \cup \{\langle 0, 0, \ldots, 0\rangle\} & \text{if } i = 1, \\ \pi_{R_i} I & \text{if } i \geq 2 \end{cases}$$

and I is the following table (relation):

A_1	A_2	$\ldots$	A_p	B_1	B_2	$\ldots$	B_q
1	0	$\ldots$	0	1	1	$\ldots$	1
0	1	$\ldots$	0	2	2	$\ldots$	2
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$
0	0	$\ldots$	1	p	p	$\ldots$	p

Let us see that $\mathbb{A}$ is locally consistent. For each $i \neq 1$, each tuple $u \in I_i$ has a matching tuple $v \in I_j$: u is a projection of a tuple $t \in I$ on R_i, so if we project t on R_j we will get a matching tuple v. Similarly, for $u \in I_1 \setminus \{\langle 0, 0, \ldots, 0\rangle\}$. To find a matching tuple for $\langle 0, 0, \ldots, 0\rangle$, observe that R_j is not contained in R_1, by claim (i). Hence, $A_r \notin R_j$ for some r, and so $\langle 0, 0, \ldots, 0, r, r, \ldots, r\rangle \in I_j$ is a matching tuple.

To show that $\mathbb{A}$ is not globally consistent, we prove that $\langle 0, 0, \ldots, 0\rangle \notin \pi_{R_1}(\bowtie_i I_i)$. Suppose that $\langle 0, 0, \ldots, 0, k_1, k_2, \ldots, k_q\rangle$ is the matching tuple in $\pi_{R_1}(\bowtie_i I_i)$. By (ii), A_1 is used in some relation R_j with $j \geq 2$, say R_{j_1}. It follows that

$$k_i \neq 1 \text{ whenever } B_i \in R_{j_1}\,. \tag{*}$$

We will show that $k_i \neq 1$ for all i. Since $\mathcal{R}'$ is connected, for each i' there is a path $R_{j_1}, R_{j_2}, \ldots, R_{j_m}$ in over vertices $B_1, B_2, \ldots, B_q$ such that $i' \in R_{j_m}$. The property (*) propagetes along this path and ultimatly gives $k_{i'} \neq 1$.

Identical argument shows that $k_i \neq s$ for all i and $s = 2, 3, \ldots p$. Hence, the matching tuple cannot exist, and thus $\mathbb{A}$ shows that $\mathcal{R}$ violates (3), which is a contradiction.

$\mathcal{R}'$ is disconnected. In this case we will find a contradiction with claim (iii) showing that $\mathcal{R}$ must contain an articulation set. Choose one component and let $S_1, S_2, \ldots, S_k \in \mathcal{R}$ be the involved relations. Let $S = S_1 \cup S_2 \cup \cdots \cup S_k$ and $R_1' = R_1 \cap S$.

If $R_1' \subseteq S_i$ for some i, $R_1 \cap S_i$ is an articulation set for $\mathcal{R}$, which is a contradiction. Hence, we can assume that $R_1 \not\subseteq S_i$ for all i. It follows that

$$\mathcal{R}|_S = \{R_1', S_1, S_2, \ldots, S_k\}$$

is reduced. By Lemma 3, $\mathcal{R}|_S$ satisfies (3), so by minimality of R, $\mathcal{R}|_S$ is acyclic. As it is reduced, connected, and has at least two edges, it must have an articulation set itself.

Suppose $Z = S_i \cap S_j$ is an articulation set in $\mathcal{R}|_S$. We claim it is also an articulation set in $\mathcal{R}$, i.e., $\mathcal{R}|_{Z^{\complement}}$ is not connected (by $Z^{\complement}$ we mean attributes of $\mathcal{R}$ not in Z). Pick a vertex $x \in R_1'$ and a vertex y in some other connected component of $(\mathcal{R}|_S)|_{S\setminus Z}$. Suppose there is a path from y to x in $\mathcal{R}|_{Z^{\complement}}$. We will turn it into a path in $(\mathcal{R}|_S)|_{S\setminus Z}$ and thus obtain a contradiction. Let T be the first edge of this path outside of $\{S_1, S_2, \ldots, S_n\}$ (if there is none we are done). It has a common vertex $v \notin Z$ with the previous edge, say S_ℓ. Suppose that $T \neq R_1$. Since $\{S_1, S_2, \ldots, S_k\}$ exhaust a connected component of $\mathcal{R}'$, T can only intersect with S_ℓ within R_1. It follows that $v \in R_1'$, which gives a path in $(\mathcal{R}|_S)|_{S\setminus Z}$.

For an articulation set $Z = S_i \cap R_1'$, an analogous proof shows that $S_i \cap R_1$ is an articulation set in $\mathcal{R}$. This completes the case of disconnected $\mathcal{R}'$. □

Proof of (4) ⇒ (1). We show by induction on the number of vetices that each acyclic reduced graph with at least two hyperedges has at least two ears.

If $\mathcal{F}$ is acyclic reduced and has at least two edges, it must contain an articulation set $f \cap f'$. Then we can split $\mathcal{F}$ into $\mathcal{F}_1$ and $\mathcal{F}_2$ such that hyperedges

from $\mathcal{F}_1$ and $\mathcal{F}_2$ do not intersect outside of $f \cap f'$. Clearly $\mathcal{F}_1$ and $\mathcal{F}_2$ are acyclic and reduced.

If $f, f' \in \mathcal{F}_i$, then each ear in $\mathcal{F}_i$ is an ear in $\mathcal{F}$. Indeed, take an ear e with a witness w. For any $g \in \mathcal{F}_{i-1}$, $e \cap g \subset f \cap f'$. If $e \neq f$, we have $e \cap g \subseteq e \cap f \subseteq w$; if $e = f$, $e \cap g \subseteq e \cap f' \subseteq w$. By induction hypothesis, $\mathcal{F}_i$ has at least two ears, and by the argument above both are ears in $\mathcal{F}$.

The remaining case is that f, f' are split between $\mathcal{F}_1$ and $\mathcal{F}_2$, say $f \in \mathcal{F}_1$, $f' \in \mathcal{F}_2$. If $\mathcal{F}_1$ is a singleton, then f is clearly an ear in $\mathcal{F}$ with a witness f'. If $\mathcal{F}_1$ has at least two edges, it has two ears by induction the hypothesis. At least one ear e is different from f. One shows that e is an ear in $\mathcal{F}$ just like in the first case. Similarly one finds at least one ear of $\mathcal{F}$ in $\mathcal{F}_2$. □

Exercises

1. Show claims (i) – (iii) in the proof of Theorem 6.
   (i) Follows by Lemma 3 (2) and Lemma 5 (2).

   (ii) Suppose A occurs only in S. Show that $\mathcal{R}' = \mathcal{R}|_{\{A\}^\complement}$ is a smaller counterexample using Lemma 3 (1) and the following observation: if U is a cycle in $\mathcal{R}$, then $U' = U \backslash \{A\}$ is a cycle in $\mathcal{R}'$. Indeed, $\mathcal{R}'|_{U'}$ is clearly connected, and it has more than one edge after reducing, because the original cycle has at least 3 edges (there is no cycle with two edges only). As intersections of two edges in U' are the same as in U, each articulation set in U' is an articulation set in U.

   (iii) Suppose that $\mathcal{R}$ has an articulation set Z. Then one can split relations of $\mathcal{R}$ into two nonempty subsetes, $\mathcal{R}_1$ and $\mathcal{R}_2$ such that elements from $\mathcal{R}_1$ and $\mathcal{R}_2$ do not intersect outside of Z. Let U_1, U_2 be the sets of attributes used in $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively. Note that $U_1 \cap U_2 = Z$. Let W be a cycle in $\mathcal{R}$. Since W has no articulation set, $W \subseteq U_1$ or $W \subseteq U_2$. By Lemma 3, either $\mathcal{R}|_{U_1}$ or $\mathcal{R}|_{U_2}$ contradicts the minimality of $\mathcal{R}$.

2. $\mathcal{R}$ has *running intersection* property if there is an ordering $R_1, R_2, \ldots, R_n$ of elements of $\mathcal{R}$ such that for $2 \leq i \leq n$ there exists $j_i < i$ such that $R_i \cap (R_1 \cup \cdots \cup R_{i-1}) \subseteq R_{j_i}$. In other words, the intersection of R_i with the previous R_j's is contained in one of these. Prove that $\mathcal{R}$ has running intersection property iff $\mathcal{R}$ is acyclic.

3. A *Berge cycle* in a hypergraph $\mathcal{F}$ is a sequence $f_1, v_1, f_2, v_2, \ldots, f_n, v_n, f_{n+1}$ such that: (1) v_i are distinct vertices of $\mathcal{F}$; (2) f_i are distinct edges of $\mathcal{F}$ and $f_1 = f_{n+1}$; (3) $n \geq 2$; (4) $v_i \in f_i \cap f_{i+1}$ for all i. A hypergraph is *Berge acyclic* if it does not contain a Berge cycle.

   Prove that Berge acyclicity is sufficient but not necessary for acyclicity.

   Prove that any hypergraph in which two edges have two nodes in common is Berge cyclic.

7 Negation

Class 5

Relational algebra

Full relational algebra (RA) is obtained by extending the SPC algebra with the operations $\cup$ and $-$ as follows:

- If E, E' are expressions of RA of the same arity, then $E \cup E'$ and $E - E'$ are expressions as well. The semantics is the usual set-theoretic union and difference.

Non-recursive Datalog with negation

The rules of nr-Datalog$^\neg$ are of the form

$$q: \qquad S(u) \leftarrow L_1, L_2, \ldots, L_n$$

where S is a fresh relation name, L_i are *literals*, i.e., either $R(v)$ or $\neg R(v)$ for some relation name R (different from S) and a tuple of variables and/or constants v. As usually, we need a *safety restriction* to prevent infinite answers: we assume that each variable from u is used in some *positive* literal L_i, i.e., a literal of the form $R(v)$.

The semantics is given as

$$q^{\mathbb{A}} = \left\{\theta(u) \;\middle|\; \theta\colon \mathrm{var}(q) \to \mathcal{D} \wedge \forall_i (L_i = R(v_i) \implies \theta(v_i) \in R^{\mathbb{A}} \wedge L_i = \neg R(v_i) \implies \theta(v_i) \notin R^{\mathbb{A}})\right\}.$$

Equality can be added without increasing expressivity.

A *program* is a sequence of rules

$$S_1 \leftarrow \mathit{body}_1; \quad S_2 \leftarrow \mathit{body}_2; \quad \ldots \quad S_m \leftarrow \mathit{body}_m;$$

such that $S_i \notin \mathcal{R}$ and $S_i \notin \mathit{body}_j$ for $j \le i$ (i.e., the program is non-recursive). Note that we do not assume that all S_i are distinct. The semantics is defined recursively: $S_i^{\mathbb{A}} = \bigcup_{j=\ell_i}^{k_i} q_j^{\mathbb{A}}$ where $q_{\ell_i}, q_{\ell_i+1}, \ldots, q_{k_i}$ are all the rules with S_i in their head.

Consider the following programs:

$$\begin{aligned}
P_1: \quad \mathit{HitchActor}(z) &\leftarrow \mathit{Movie}(_, \text{'Hitchcock'}, z)\\
\mathit{NotAns1}(x) &\leftarrow \mathit{Movie}(x, _, z), \neg \mathit{HitchActor}(z)\\
\mathit{Ans1}(x) &\leftarrow \mathit{Movie}(x, _, _), \neg \mathit{NotAns1}(x)\\
P_2: \quad \mathit{NotAns2}(x) &\leftarrow \mathit{Movie}(x, _, z), \neg \mathit{Movie}(x', \text{'Hitchcock'}, z),\\
&\qquad \mathit{Movie}(x', \text{'Hitchcock'}, _)\\
\mathit{Ans2}(x) &\leftarrow \mathit{Movie}(x, _, _), \neg \mathit{NotAns2}(x)
\end{aligned}$$

Are they equivalent? If not, which one expresses the query "Which movies feature only actors that have played in a movie directed by Hitchcock?"

Lemma 6. *Relation algebra and non-recursive Datalog with negation are equivalent.*

Proof. Easy. □

Relational calculus

Relational calculus is simply first order logic on purely relational structures (with constants). The formulae use the connectives $\wedge, \vee, \neg, \implies, \iff$, the quantifiers $\exists, \forall$, relation symbols from the schema $\mathcal{R}$, as well as $=$ and $\neq$. Relational calculus queries are formed as

$$\{(u_1, u_2, \ldots, u_n) \mid \varphi\}$$

where $\mathrm{var}(\bar{u}) = \mathrm{var}(\varphi)$. E.g.,

$$\{t \mid \exists a\ \mathit{Movie}(t, \text{'Hitchcock'}, a) \wedge \neg \mathit{Movie}(t, \text{'Hitchcock'}, \text{'Hitchcock'})\}$$

$$\{t \mid \exists a, d\ \mathit{Movie}(t, d, a) \wedge \forall a' \left[\exists d' \mathit{Movie}(t, d', a') \implies \exists t' \mathit{Movie}(t', \text{'Hitchcock'}, a')\right]\}$$

The central issue in the definition of the semantics of relational calculus queries is safety. Consider the following three queries.

$$\begin{aligned} q_1 &= \{x \mid \neg \mathit{Movie}(\text{'Avatar'}, \text{'Cameron'}, x)\}\ , \\ q_2 &= \{(x, y) \mid \mathit{Movie}(\text{'Avatar'}, \text{'Cameron'}, x) \vee \mathit{Movie}(y, \text{'Cameron'}, \text{'Winslet'})\}\ , \\ q_3 &= \{x \mid \forall y, z\ \mathit{Movie}(x, y, z)\}\ . \end{aligned}$$

Under the standard semantics the queries q_1, q_2 give infinite answers and q_3 is always empty. One could try to fix a particular domain, but then the answer would depend on this domain. The solution is to concentrate on domain-independent queries. (None of the above three is domain independent.)

Given a query Q and an instance I over $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ we evaluate Q on I *relative to* U, $\mathrm{adom}(I) \subseteq U \subseteq \mathcal{D}$, by considering the classical interpretation in the structure

$$\langle U, I\rangle = \langle U, R_1^I, R_2^I, \ldots, R_n^I\rangle$$

where the relations $R_1^I, R_2^I, \ldots, R_n^I$ are given by I.

In general, the answer $Q^{\langle U,I\rangle}$ depends on U, e.g. $U = \mathrm{adom}(I)$ gives the active domain semantics, and $U = \mathcal{D}$ gives the classical semantics. We concentrate on queries for which the choice of U does not matter. A query Q is *domain independent* iff for each instance I, and all U, U' such that $\mathrm{adom}(I) \subseteq U \subseteq \mathcal{D}$, $\mathrm{adom}(I) \subseteq U' \subseteq \mathcal{D}$ it holds that $Q^{\langle U,I\rangle} = Q^{\langle U',I\rangle}$.

Let $\mathrm{CALC}_{\mathrm{di}}$ denote the queries that can be expressed by domain independent relational calculus queries, and let $\mathrm{CALC}_{\mathrm{adom}}$ denote those that can be expressed by arbitrary relational calculus queries under the active domain semantics.

Lemma 7. $\mathrm{CALC}_{\mathrm{di}} \subseteq \mathrm{CALC}_{\mathrm{adom}}$.

Proof. If Q is domain independent, it yields the same answer under the active domain semantics and under any relative interpretation. □

Theorem 7. $\mathrm{CALC}_{\mathrm{di}} \equiv \mathrm{CALC}_{\mathrm{adom}} \equiv \mathrm{RA} \equiv$ *nr-Datalog*$^\neg$.

Proof. It remains to see that $\mathrm{RA} \subseteq \mathrm{CALC}_{\mathrm{di}}$ and $\mathrm{CALC}_{\mathrm{adom}} \subseteq \mathrm{RA}$. To prove the first containment, for each $E \in \mathrm{RA}$ we construct equivalent $\{\bar{x} \mid \varphi_E\} \in \mathrm{CALC}_{\mathrm{di}}$ by induction.

- If $E = R \in \mathcal{R}$, let $\varphi_E = R(\bar{x})$.
- If $E = \{(u_1, u_2, \ldots, u_n)\}$, let $\varphi_E = x_1 = u_1 \wedge x_2 = u_2 \wedge \cdots \wedge x_n = u_n$.
- If $E = \sigma_F E'$, let $\varphi_E = \varphi_{E'} \wedge \psi_F$, where ψ_F corresponds to the selection condition.
- If $E = \pi_{i_1, i_2, \ldots, i_j} E'$, let $\varphi_E = \exists y_1, y_2, \ldots, y_n \; x_1 = y_{i_1} \wedge x_2 = y_{i_2} \wedge \cdots \wedge x_j = y_{i_j} \varphi_{E'}(\bar{y})$.
- If $E = E_1 \times E_2$, let $\varphi_E = \varphi_{E_1} \wedge \varphi_{E_2}(x_{p+1}, x_{p+2}, \ldots, x_{p+q})$, where $p = \mathrm{ar}(E_1)$, $q = \mathrm{ar}(E_2)$.
- If $E = E_1 \cup E_2$, let $\varphi_E = \varphi_{E_1} \vee \varphi_{E_2}$.
- If $E = E_1 - E_2$, let $\varphi_E = \varphi_{E_1} \wedge \neg \varphi_{E_2}$.

It is straightforward to check that at each step we preserve domain independence.

For the second containment one similarly constructs for each formula φ an equivalent RA expression E_φ. First let $E_{\mathrm{adom}} = \bigcup_{i,j} \pi_i R_j$. This query computes the unary relation containing the active domain of the given instance.

Next, proceed by induction on the structure of the formula. Let us consider a couple of examples.

- If $\varphi(y_1, y_2, \ldots, y_m) = \psi(t_1, t_2, \ldots, t_n)$, let $E_\varphi = \pi_{\bar{\ell}} \sigma_F R$ with suitably chosen F and $\bar{\ell}$.
- If $\varphi = y_1 = y_2$, let $E_\varphi = \sigma_{1=2}(E_{\mathrm{adom}} \times E_{\mathrm{adom}})$.
- If $\varphi = \psi_1(y_1, y_2) \vee \psi_2(y_2, y_3)$, let $E_\varphi = E_{\psi_1} \times E_{\mathrm{adom}} \cup E_{\mathrm{adom}} \times E_{\psi_2}$.
- If $\varphi = \neg \psi(y_1, y_2)$, let $E_\varphi = E_{\mathrm{adom}} \times E_{\mathrm{adom}} \times E_{\mathrm{adom}} - E_\psi$.

The remaining cases are very similar. □

Static analysis of CALC queries

Conjunctive queries are much less expressive that CALC but they have one major advantage: decidable containment, and minimization.

Theorem 8. *Satisfiability of CALC queries is recursively enumerable, but not decidable.*

Proof. See proof of Theorem 6.3.1 in [1]. □

Corollary 4.

- *Equivalence and containment of CALC queries are co-r.e., but not decidable.*
- *Domain independence of CALC queries is co-r.e., but not decidable.*

Proof. Good exercise. □

Exercises

1. Let SPCU$^{\neq}$ denote SPCU algebra extended with inequality in selection formulae: $\sigma_{i\neq j}E$, $\sigma_{i\neq a}E$. Determine containment relations between SPCU, SPCU$^{\neq}$, and RA.
   $\sigma_{i\neq j}$ and $\sigma_{i\neq a}$ are monotonic.
2. The *division* operator, denoted $:$, is added to the named algebra as follows. For relations I and J with $\text{sort}(J) \subseteq \text{sort}(I)$, the value of $I : J$ is the set of tuples $t \in \pi_A I$, where $A = \text{sort}(I) \setminus \text{sort}(J)$, such that $(\{t\} \times J) \subseteq I$. Show that relational algebra can simulate the division operator.
   Solution: $I : J = \pi_A I - \pi_A(\pi_A((\pi_A I) \times J - I))$.

Complexity and expressive power of the relational calculus

Class 6

I just state the key results. The relevant sections from [1] are 17.1 and 17.2.

Theorem 9. *CALC queries can be evaluated in LOGSPACE.*

This means that for each query there is a Turing Machine with read-only input tape, and a working tape that computes the query on every input using only logarithmically many cells of the working tape. In fact, a stronger theorem can be proved.

Theorem 10. *CALC queries can be evaluated in* AC_0.

This means that for each query there exists a constant depth boolean circuit consisting of polynomially many gates of unbounded fan-in. In the context of parallel computation, this tells us that CALC queries can be evaluated in constant time using polynomially many processors.

Thus CALC is a very efficient language. Unfortunately, we pay for this in the expressivity.

Theorem 11. *The following queries cannot be expressed in CALC:*

- *The cardinality of (unary) U is even.*
- *The binary relation E is connected.*

In particular it is not possible to express transitive closure of a given relation. This is a major weakness of CALC, and motivates extensions discussed in the next section.

8 Recursion

Class 7

Adding recursion to CALC can be done in two different ways. The following example illustrates the difference.

Recall the *Game of Life*. Assume it is played on a finite undirected graph. Each node is a cell. A cell can be either dead or alive. If a cell is alive, it can be red or blue (or both). For a start, assume that the only rule of the evolution of the system is

(+) if the cell is dead and has two neighbors of the same color, it becomes alive and inherits this color.

Note that with this rule the system finally becomes stable, no matter what the initial configuration is. Furthermore, the number of steps needed before getting to this state is linear in the size of the graph.

Now, add the following rule:

(−) if the cell has more then three alive neighbors, it becomes dead.

In the presence of both rules, it is possible that the system will evolve forever. A good exercise is to provide a graph and an initial configuration such that some states will change their colors infinitely many times. Note that since there are only finitely many possible configurations, the system will start oscillating at some point: the first time a configuration repeats, the part of the history between those occurrences will repeat forever.

Under the first rule only, we observe *inflationary* evolution of the system. Under both rules, the system is *non-inflationary*. In the next subsection we will see how these intuitions can be applied to extend the relational algebra with recursion.

WHILE queries

It is easy to see that RA can be extended with assignment := and sequential composition ; without increasing expressivity (by the virtue of compositionality). The real step out of RA is adding looping. We will work with the following programs

$$\begin{aligned} P ::=&\ T := E \mid P; P \mid \texttt{while change do } P \texttt{ end} \\ E ::=&\ E \times E \mid \sigma E \mid \pi E \mid E \cup E \mid E - E \end{aligned}$$

Note that in our programs there is no explicit termination condition: the loop runs until it reaches a stable state, i.e., as long as the contents of the tables changes.

Assume G stores the edge relation of a graph. The following program com-

putes the transitive closure of G.

$T := G;$
`while change do`
  $T := T \cup \pi_{1,4}\, \sigma_{2=3}\, (T \times G);$
`end`

Note that the program terminates: it can make at most quadratic number of steps.

The following program does not terminate on every inputs (as an exercise, find one).

$T := G;$
`while change do`
  $ToRemove := \{(x, y) \mid \exists z\, T(x, z) \wedge T(z, y)\};$
  $ToAdd := \{(x, y) \mid \exists z\, \neg T(x, z) \wedge \neg T(z, x) \wedge \neg T(z, y) \wedge \neg T(y, z)\};$
  $T := (T \cup ToAdd) - ToRemove;$
`end`

Proposition 3. *The halting problem for WHILE programs is undecidable.*

Proof. A simple reduction from RA satisfiability. □

WHILE$^+$ queries

WHILE queries follow the non-inflationary approach. They give more freedom, but do not guarantee termination. Let us now see an inflationary variant.

WHILE$^+$ queries are defined just like WHILE queries, with incremental assignment $+=$ instead of the usual assignment $:=$. Thus we can only write $T \mathrel{+}= E$, which translates as $T := T \cup E$. Initially, all auxiliary relations (i.e., relations defined by the program) are assumed empty.

The transitive closure program shown above is actually a WHILE$^+$ program:

$T \mathrel{+}= G;$
`while change do`
  $T \mathrel{+}= \pi_{1,4}\, \sigma_{2=3}\, (T \times G);$
`end`

As pointed out in the Game of Life example, under inflationary evolution, every system reaches a stable state. Hence WHILE$^+$ programs always terminate.

A more complex query that can be written in WHILE$^+$ is "pairs of nodes at even distance", where *distance* between two nodes is the length of the shortest path between them. The query is computed by the following program.

$P \mathrel{+}= \sigma_{1=2} E$;
$N \mathrel{+}= E - P$;
`while change do`
  $P \mathrel{+}= \pi_{1,4}\sigma_{2=3}(N \times E) - N$;
  $N \mathrel{+}= \pi_{1,4}\sigma_{2=3}(P \times E) - P$;
`end`;
$Answer \mathrel{+}= P$;

Note how this program makes use of difference operator. It uses difference in expressions on the right hand of asignments, but nothing is ever removed from the auxiliary relations.

Partial fixpoints: CALC $+\mu$

Let us start with an example. Assuming G stores the edge relation of a graph, let J_n denote the pairs of nodes at distance $\leq n$. It can be defined recursively as

$$\begin{aligned} J_0 &= \emptyset \\ J_n &= \varphi(J_{n-1}) \end{aligned}$$

with an FO formula

$$\varphi(x,y) = G(x,y) \vee T(x,y) \vee \exists z\, T(x,z) \wedge G(z,y)\,.$$

For each input G, the sequence J_n converges. The limit $\lim_{n\to\infty} J_n$, denoted $\mu T.\varphi$ is the transitive closure of G.

It can happen for some φ that the limit does not exist. For instance, for $\varphi = (x = 0 \wedge \neg T(0) \wedge \neg T(1)) \vee (x = 0 \wedge T(1)) \vee (x = 1 \wedge T(0))$ we have

$$\emptyset \to \{(0)\} \to \{(1)\} \to \{(0)\} \to \{(1)\} \to \{(0)\} \to \dots\,.$$

Roughly speaking, such formulae correspond to looping WHILE programs:

$T := \{(0)\}$;
`while change do`
  $T := \{(0),(1)\} - T$;
`end`

In general, *partial fixpoint operator* is defined as follows. Let $T \notin \mathcal{R}$ be a fresh relation symbol of arity m. Let $\varphi \in$ CALC be a formula over $\mathcal{R} \cup \{T\}$ with m free variables. Given a database I, define $\mu T.\varphi$ as the limit of the sequence below (provided it exists)

$$\begin{aligned} J_0 &= \emptyset\,, \\ J_n &= \varphi(J_{n-1})\,, \end{aligned}$$

where $\varphi(J_{n-1})$ is to be understood as the output of the query φ on (I, J_{n-1}). If the limit does not exists, $\mu T.\varphi$ is not defined.

The logic CALC+μ is obtained by extending CALC with the partial fixpoint operator μ. This means that $\mu T.\varphi$ can be used as a subformula

$$(\mu T.\varphi)(y_1, y_2, \ldots, y_m)\,.$$

The semantics is simply bottom-up evaluation. If some subformula has undefined semantics, so does the whole formula.

CALC+μ queries are written as

$$\{(u_1, u_2, \ldots, u_n) \mid \xi\}\,,$$

where $\mathrm{var}(\bar{u}) = \mathrm{var}(\xi)$. For instance, if φ is the formula from the transitive closure example, $\{(x, y) \mid \neg(\mu T.\varphi)(x, y)\}$ returns all pairs of nodes which are not connected with a path.

Inflationary fixpoints: CALC+μ^+

Inflationary fixpoint operator, $\mu^+ T.\varphi$, computes the limit of the sequence

$$\begin{aligned} J_0 &= \emptyset\,, \\ J_n &= J_{n-1} \cup \varphi(J_{n-1})\,. \end{aligned}$$

Observe that since

$$J_0 \subseteq J_1 \subseteq J_2 \subseteq \cdots \subseteq (\mathrm{adom}\, I)^m$$

the limit always exists. Note that the transitive closure formula φ yields the same result under both fixpoint operators: $\mu T.\varphi = \mu^+ T.\varphi$.

Remark. A different approach to guarantee that all formulae have defined meaning in every database is to demand that φ is monotone: $T \subseteq T' \implies \varphi(T) \subseteq \varphi(T')$. Unfortunately, checking if a formula is monotone is undecidable. A syntactic condition sufficient (but not necessary) for monotonicity is that the relation T is used *positively*, i.e., under an even number of negations only. It turns out that these two approaches give languages equivalent to CALC+μ^+.

Fixpoints = loops

Theorem 12. CALC + $\mu^+ \equiv$ WHILE$^+$.

Proof. For simplicity, we assume that the query is of the form $q = \{(x_1, x_2, \ldots, x_m) \mid \xi(\bar{x})\}$. We will construct a program $P_\xi \in$ WHILE$^+$ such that at the end of the computation R_ξ will store the output of the query q. We proceed by structural induction.

As the induction base consider the case where ξ is a CALC formula. The corresponding program P_ξ is simply

$$R_\xi \mathrel{+}= E_\xi;$$

where E_ξ is an RA expression equivalent to ξ.

Now, assume that $\xi = (\mu Q.\varphi)(u_1, u_2, \ldots, u_n)$. The program P_ξ should be defined as

$$
\begin{array}{l}
Q \mathrel{+}= \emptyset; \\
\texttt{while change do} \\
\quad P_\varphi; \\
\quad \{R_\varphi \texttt{ contains } \varphi(Q)\} \\
\quad Q \mathrel{+}= R_\varphi; \\
\texttt{end} \\
R_\xi \mathrel{+}= \pi_{\bar{\ell}} \sigma_F(Q);
\end{array}
$$

where $\bar{\ell}$ and F correspond to constants and variables among u_i.

Finally, if ξ is obtained by first order operations from k formulas $\xi_1, \xi_2, \ldots, \xi_k$ having μ^+ as root, $\xi = \varphi(T_1, T_2, \ldots, T_k)[T_1 := \xi_1, T_2 := \xi_2, \ldots, T_k := \xi_k]$. Let E_φ be the RA expression equivalent to φ. Define P_ξ as

$$
\begin{array}{l}
P_{\xi_1}; P_{\xi_2}; \ldots; P_{\xi_k}; \\
R_\xi \mathrel{+}= E_\varphi(R_{\xi_1}, R_{\xi_k}, \ldots, R_{\xi_k});
\end{array}
$$

The translation from WHILE$^+$ to CALC+μ^+ is left as an exercise. □

Theorem 13. WHILE $\equiv$ CALC $+ \mu$.

Proof. Translation from CALC $+ \mu$ to WHILE is a straightforward modification of the inflationary case. For the converse translation the main difficulty is to simulate sequential composition. This is done by tagging the tuples (in an extra column). We only consider an example here:

$$
\begin{array}{l}
T := G; \\
\texttt{while change do} \\
\quad ToRemove := \{(x,y) \mid \exists z\, T(x,z) \wedge T(z,y)\}; \\
\quad ToAdd := \{(x,y) \mid \exists z\, \neg T(x,z) \wedge \neg T(z,x) \wedge \neg T(z,y) \wedge \neg T(y,z)\}; \\
\quad T := (T \cup ToAdd) - ToRemove; \\
\texttt{end}
\end{array}
$$

An equivalent CALC $+ \mu$ formula is $\{(x,y) \mid (\mu Q.\varphi)(x,y,0)\}$, where φ is

$$
\begin{array}{l}
\neg Q(1,1,1) \wedge [(G(x,y) \wedge z = 0) \vee x = y = z = 1] \vee \\
\vee\, Q(1,1,1) \wedge [x = y = z = 1 \vee \\
\qquad \vee (z = 0 \wedge Q(x,y,0) \wedge \neg \exists w Q(x,w,0) \wedge Q(w,y,0)) \vee \\
\qquad \vee (z = 0 \wedge \exists w \neg Q(x,w,0) \wedge \neg Q(w,x,0) \wedge \neg Q(y,w,0) \wedge \neg Q(w,y,0))] .
\end{array}
$$

□

Complexity and expressive power

Theorem 14. *In terms of data complexity*

- *FIXPOINT queries can be evaluated in PTIME,*
- *WHILE queries can be evaluated in PSPACE.*

Proof. At every step of the computation, the intermediate results have polynomial size. In FIXPOINT computations at each step we add at least one tuple to the intermediate results, so we can only make polynomially many steps. In WHILE computations we can add and remove tuples, but still we only need polynomial space to perform the computations. □

In general, there is no equality in the theorem above: the parity of a relation is clearly a PTIME problem, but it lies beyond the expressive power of both query languages.

Theorem 15. *Neither FIXPOINT nor WHILE can express that a (unary) relation U has even cardinality.*

Proof. See proof of Theorem 17.3.2. in [1] □

It turns out however that the equality is very close: it is enough to assume that databases are *ordered*. By this we mean that $\mathcal{R}$ contains a special binary relation symbol *succ* and we only consider those databases in which *succ* table stores a proper successor relation on the whole of *adom*.

Theorem 16.

- *FIXPOINT $\equiv$ PTIME on ordered databases.*
- *WHILE $\equiv$ PSPACE on ordered databases.*

Proof. See the proof of Theorem 17.4.2 in [1] □

The equivalences in the theorem above should be understood as follows: for every PTIME Turing machine M taking as input (a reasonable encoding of) a database and returning (a reasonable encoding of) a set of tuples, there is a FIXPOINT query Q such that $M(enc(I)) = enc(Q^I)$ for every database I, and *vice versa*. Similarly for the second claim.

From the theorem above, we obtain hardness in the general (unordered) case.

Corollary 5. *In terms of data complexity*

- *FIXPOINT query evaluation is PTIME-complete,*
- *WHILE query evaluation is PSPACE-complete.*

Exercises

1. Consider the following simple game played by Adam and Eve on a finite directed graph whose vertices are divided between the players. The game starts with a token being placed in a starting position. Next, the players move the token along the edges of the graph: the next position is chosen by the owner of the current position. Eve wins the play (finite or infinite) iff at some point the token reaches a designated target vertex, or the token gets trapped in a vertex belonging to Adam with no outgoing edges. Otherwise the play is won by Adam.

   Assume the arena is represented by relations: *Move*, storing edges between positions, *Adam*, storing Adam's positions, and *Eve*, storing Eve's positions. Write a FIXPOINT (i.e., WHILE$^+$ or CALC+μ^+) query that returns pairs of vertices (v, w) such that Eve has a winning strategy in the game described above with the starting position v and the target position w.

   Is there an equivalent CALC query?

2. Show that WHILE $\equiv$ CALC on input databases with unary relations only (the program can use relations of arbitrary arity).

3. Prove that WHILE = PSPACE on ordered databases.

4. Prove that WHILE = CALC+μ.

9 Więzy w praktyce

Class 9

Niech Σ będzie pewnym zbiorem własności baz danych, sformalizowanych w w jakimś formalizmie, np. logice pierwszego rzędu. Zależności takie będziemy nazywali więzami. Będziemy rozważali wyłącznie takie bazy danych $\mathbb{A}$, że $\mathbb{A} \models \Sigma$. Wprowadzenie więzów do modelu ma liczne powody:

- *Usprawnienie projektowania schematu bazy danych.* Dzięki więzom łatwiej jest wykrywać potencjalne anomalie przy wstawianiu bądź usuwaniu wartości, czy też źródła redundancji.

- *Ochrona spójności danych.* Akceptujemy bądź odrzucamy transakcję w zależności od tego, czy modyfikacja jest spójna z więzami, czy nie.

- *Efektywnie składowanie.* Więzy mogą podpowiedzieć, jak podzielić dane na podtable i jak skonstruować indeksy.

- *Optymalizacja zapytań.* Być może zapytanie może być krótsze, jeśli ma działać tylko na danych spełniających więzy. Być może można je sprytniej obliczyć.

Zależności funkcyjne

Zależność funkcyjna (ang. *functional dependency*) dla relacji $R \in \mathcal{R}$ to wyrażenie postaci

$$R : \ X \to Y$$

dla pewnych zbiorów atrybutów $X, Y \subseteq \mathrm{sort}(R)$, odczytywane „$X$ determinuje Y". Jeśli $Y = \mathrm{sort}(R)$, to X jest *kluczem* (ang. *key*) w relacji R. Jeśli dodatkowo $|X| = 1$, to X jest *kluczem głównym* (ang. *primary key*). Zwykle będziemy pomijali R w napisie powyżej.

Niech I będzie instancją relacji R. Mówimy, że I spełnia $X \to Y$, oznaczane $I \models X \to Y$, jeśli

$$\forall\, s, t \in I \quad \pi_X(s) = \pi_X(t) \implies \pi_Y(s) = \pi_Y(t)\,.$$

Dla przykładu, w dawniejszych czasach baza kin spełniałaby zależność

$$\mathit{Kino}, \mathit{Godzina} \to \mathit{Tytuł}\,,$$

a w zamyśle twórców systemu PESEL,

$$\mathit{PESEL} \to \mathit{Imie}, \mathit{Nazwisko}, \mathit{DataUrodzenia}, \mathit{NumerButa}\,.$$

Fakt 1. *Niech I będzie instancją relacji o zbiorze atrybutów U spełniającą $I \models X \to Y$. Dla $Z = U - XY$ zachodzi*

$$I = \pi_{XY}(I) \bowtie \pi_{XZ}(I)\,.$$

Dowód. Inkluzja $\subseteq$ jest oczywista. Pokażemy inkluzję $\supseteq$. Weźmy $r = \pi_{XY}(s) \bowtie \pi_{XZ}(t)$ dla pewnych $s, t \in I$. Chcemy pokazać, że $r \in I$. Skoro s, t są złączalne, to $\pi_x(s) = \pi_X(t)$. Na mocy zależności $X \to Y$ wnioskujemy, że $\pi_Y(s) = \pi_Y(t)$. Zatem $\pi_{XY}(s) = \pi_{XY}(t)$. Stąd mamy $\pi_{XY}(t) = \pi_{XY}(s) = \pi_{XY}(r)$ oraz $\pi_Z(t) = \pi_Z(r)$, czyli $r = t \in I$. □

Semantyczna implikacja

Niech Σ, Γ będą zbiorami zależności funkcyjnych (ogólniej: więzów) nad $\mathcal{R}$. Mówimy, że Σ (semantycznie) implikuje Γ, ozn. $\Sigma \models \Gamma$, jeśli dla każdej bazy danych $\mathbb{A}$ nad $\mathcal{R}$,

$$\mathbb{A} \models \Sigma \implies \mathbb{A} \models \Gamma\,.$$

Na przykład dla $\Sigma_1 = \{A \to C, B \to C, CD \to E\}$, nad $\mathcal{R} = \{R(ABCDE)\}$, mamy $\Sigma_1 \models AD \to E$ oraz $\Sigma_1 \models CDE \to C$. W tym drugim przypadku, tak naprawdę zachodzi $\emptyset \models CDE \to C$.

Rozstrzyganie, czy dla zadanych zbiorów więzów Σ, Γ zachodzi $\Sigma \models \Gamma$ jest jednym z najważniejszych problemów omawianej przez nas teorii (również z przyczyn praktycznych). Dla zbioru zależności funkcyjnych Σ zdefiniujmy zbiór implikowanych zależności

$$\Sigma^* = \{R : X \to Y \;\big|\; \Sigma \models R : X \to Y,\ R \in \mathcal{R},\ X, Y \subseteq \mathrm{sort}(R)\}\,.$$

Zwróćmy uwagę, że zbiór ten może mieć rozmiar wykładniczy ze względu na Σ i $\mathcal{R}$: wystarczy zauważyć, że dla $X \subseteq Y \subseteq \mathrm{sort}(R)$, $R \in \mathcal{R}$, mamy $(R : Y \to X) \in \Sigma^*$ bez względu na zawartość Σ. Mimo to, w przypadku zależności funkcyjnych, problem implikacji ma wielomianowe rozwiązanie.

Twierdzenie 1. *Dla zadanych zbiorów zależności funkcyjnych Σ, Γ można w czasie wielomianowym rozstrzygnąć, czy $\Sigma \models \Gamma$.*

Dowód. Wystarczy pokazać tezę $\mathcal{R} = \{R\}$ i $\Gamma = \{X \to Y\}$. Zdefiniujmy

$$X^* = \{A \in \mathrm{sort}(R) \mid \Sigma \models X \to A\}.$$

Na przykład dla $\Sigma_1 = \{A \to C, B \to C, CD \to E\}$ mamy $A^* = AC$, $(AB)^* = ABC$, $(AD)^* = ACDE$. Łatwo sprawdzić, że $\Sigma \models X \to Y$ wtedy i tylko wtedy, gdy $Y \subseteq X^*$. Zatem do udowodnienia twierdzenia brakuje tylko wielomianowego algorytmu obliczającego X^*.

Rozważmy następującą procedurę:

```
unused := Σ; closure := X;
while change do
  if W → Z ∈ unused and W ⊆ closure then
    unused := unused − {W → Z};
    closure := closure ∪ {W → Z};
end;
return closure;
```

Poprawność jest oczywista. Dla pokazania pełności wystarczy wskazać instancję, która będzie spełniała Σ i nie będzie spełniała żadnej zależności postaci $X \to B$ dla B nie należących do `closure`. Łatwo sprawdzić, że powyższe warunki spełnia instancja złożona z dwóch tylko krotek, s, t, takich że $\pi_A(s) = \pi_A(t) = 0$ dla $A \in$ `closure`, oraz $\pi_B(s) = 1$, $\pi_B(t) = 2$ dla $B \notin$ `closure`. □

Problem implikacji dla innych klas zależności omawiamy w dalszej części rozdziału.

Wyprowadzanie zależności

Rozważmy następujące reguły wnioskowania:

FD1 (zwrotność) jeśli $Y \subseteq X$, to $X \to Y$;

FD2 (rozszerzanie) jeśli $X \to Y$, to $XZ \to YZ$;

FD3 (przechodniość) jeśli $X \to Y$ i $Y \to Z$, to $X \to Z$.

Na przykład, ze zbioru zależności $\Sigma_1 = \{A \to C, B \to C, CD \to E\}$ możemy wywnioskować $AD \to E$ jak następuje:

1. $A \to C$ należy do Σ_1;

2. $AD \to CD$ wynika z 1. na mocy FD2;

3. $CD \to E$ należy do Σ_1;

4. $AD \to E$ wynika z 2 i 4 na mocy FD3.

Formalnie, *dowodem* zależności σ z Σ według reguł FD1, FD2, FD3 nazywamy ciąg zależności $\sigma_1, \sigma_2, \ldots, \sigma_n$ taki że $\sigma_n = \sigma$ oraz σ_i należy do Σ lub jest otrzymane z poprzednich zależności za pomocą którejś z reguł FD1, FD2, FD3. Jeśli istnieje dowód σ z Σ według reguł $\mathcal{A}$, piszemy $\Sigma \vdash_{\mathcal{A}} \sigma$.

Zestaw reguł wyprowadzenia dla klasy więzów $\mathcal{W}$ jest

- *poprawny*, jeśli $\Sigma \vdash_{\mathcal{A}} \sigma$ implikuje $\Sigma \models \sigma$ dla wszystkich $\Sigma \subseteq \mathcal{W}$ i $\sigma \in \mathcal{W}$;
- *pełny*, jeśli $\Sigma \models \sigma$ implikuje $\Sigma \vdash_{\mathcal{A}} \sigma$ dla wszystkich $\Sigma \subseteq \mathcal{W}$ i $\sigma \in \mathcal{W}$.

Twierdzenie 2. *System reguł FD1, FD2, FD3 jest poprawny i pełny dla klasy zależności funkcyjnych.*

Dowód. Poprawność jest łatwa. Pokażemy pełność. Załóżmy, że $\Sigma \models X \to Y$. Pokażemy najpierw, że $\Sigma \vdash X \to X^*$.

Będziemy postępowali przez indukcję ze względu na działanie algorytmu z dowodu Twierdzenia 1. Niech $\texttt{closure}_i$ oznacza wartość zmiennej `closure` po i obrotach pętli dla pewnego ustalonego wykonania algorytmu. Pokażemy indukcyjnie, że $\Sigma \vdash X \to \texttt{closure}_i$. Na początku mamy $\texttt{closure}_0 = X$ i dowodem dla $X \to X$ jest $\sigma_1 = X \to X$. Załóżmy, że mamy dowód $\sigma_1, \sigma_2, \ldots, \sigma_N$ dla $X \to \texttt{closure}_i$. Przypuśćmy, że algorytm wybiera w kolejnym obrocie pętli zależność $W \to Z \in \Sigma$, $W \subseteq \texttt{closure}_i$ i że $\texttt{closure}_{i+1} = \texttt{closure}_i \cup Z$. Dowód dla $X \to \texttt{closure}_{i+1}$ otrzymujemy dodając do $\sigma_1, \sigma_2, \ldots, \sigma_N$ zależności:

$$\begin{aligned}
&\sigma_{N+1} : W \to Z && \text{(należy do } \Sigma\text{)}\\
&\sigma_{N+2} : \texttt{closure}_i \to W && \text{(zwrotność)}\\
&\sigma_{N+3} : \texttt{closure}_i \to Z && \text{(przechodniość)}\\
&\sigma_{N+4} : \texttt{closure}_i \to \texttt{closure}_{i+1} && \text{(rozszerzanie)}\\
&\sigma_{N+5} : X \to \texttt{closure}_{i+1} && \text{(przechodniość)}
\end{aligned}$$

Aby uzyskać dowód dla $X \to Y$ ze skonstruowanego powyżej dowodu $\sigma_1, \sigma_2, \ldots, \sigma_K$ dla $X \to X^*$ wystarczy dopisać na końcu:

$$\begin{aligned}
&\sigma_{K+1} : X^* \to Y && \text{(zwrotność)}\\
&\sigma_{K+2} : X \to Y && \text{(przechodniość)}
\end{aligned}$$

□

Poprawny i pełny zestaw reguł wnioskowania dla klasy więzów $\mathcal{W}$ nazywa się często aksjomatyzacją dla $\mathcal{W}$. Istnienie aksjomatyzacji dla różnych klas więzów jest kolejnym kluczowym zagadnieniem teorii więzów. Do tego tematu jeszcze wrócimy.

Zależności złączeniowe

Zależność złączeniowa to wyrażenie postaci $R\colon \bowtie [X_1, X_2, \ldots, X_n]$, gdzie $X_1 \cup X_2 \cup \cdots \cup X_n = \mathrm{sort}(R)$. Podobnie jak dla zależności funkcyjnych często pomijamy R w powyższym zapisie. Jeśli I jest instancją R, to

$$I \models \bowtie [X_1, X_2, \ldots, X_n] \iff I = \bowtie_{i=1}^{n} \pi_{X_i}(I)\,.$$

Jeśli w powyższym zapisie $n = 2$, mówimy o *zależności wielowartościowej*. Zwykle używana jest notacja $X \twoheadrightarrow Y$:

$$I \models X \twoheadrightarrow Y \iff I \models \bowtie [XY, X(\mathrm{sort}(R) - Y)]\,.$$

Równoważnie można by napisać $I \models \bowtie [XY, \mathrm{sort}(R) - Y]$.

Poniższy fakt ujawnia związki między zależnościami wielowartościowymi i zależnościami funkcyjnymi.

Fakt 2. *Niech $X \cup Y \cup Z$ będzie podziałem $\mathrm{sort}(R)$ na rozłączne zbiory i niech Σ będzie zbiorem zależności funkcyjnych nad $\{R\}$. Wtedy*

$$\Sigma \models \bowtie [XY, XZ] \iff \Sigma \models X \to Y \text{ lub } \Sigma \models X \to Z\,.$$

Dowód. Implikacja $\Leftarrow$ wynika wprost z Faktu 1. Aby wykazać $\Rightarrow$ załóżmy, że warunek po prawej nie jest spełniony. Wtedy istnieją $C \in Y - X^*$ i $D \in Z - X^*$. Rozważmy $I = \{u, v\}$ dla pewnych u, v spełniających $u(A) = v(A)$ dla $A \in X^*$ i $u(B) = 0$, $v(B) = 1$ dla $B \notin X^*$. Oczywiście $I \models \Sigma$. Z drugiej strony, istnieje $w \in \pi_{XY} I \bowtie \pi_{XZ} I$ takie że $w(C) = 0$, $w(D) = 1$, czyli $w \notin I$. Więc $I \not\models \bowtie [XY, XZ]$. □

Można pokazać, że dla zależności złączeniowych nie istnieje sensowna aksjomatyzacja. (Różne pojęcia „sensowności" aksjomatyzacji będziemy rozważali w rozdziale 11.) Jeśli jednak ograniczymy się do zależności wielowartościowych, nawet w połączeniu z zależnościami funkcyjnymi, wtedy aksjomatyzacja istnieje.

W rozdziale 10 zobaczymy, że mimo wszystko implikacja dla zależności funkcyjnych i zależności złączeniowych razem jest rozstrzygalna.

Zależności inkluzji

Trzecim z popularnych rodzajów więzów są zależności inkluzji, tzn. wyrażenia postaci

$$R[A_1, A_2, \ldots, A_n] \subseteq S[B_1, B_2, \ldots, B_n]$$

dla $R, S \in \mathcal{R}$, $A_1, A_2, \ldots, A_n \in \mathrm{sort}(R)$ i $B_1, B_2, \ldots, B_n \in \mathrm{sort}(S)$.

Dla zależności inkluzji istnieje prosta aksjomatyzacja, a więc implikacja jest rozstrzygalna. Niestety nie ma z tego wiele pożytku, bo same zależności inkluzji są mało przydatne w praktyce. Najczęściej stosuje się je w połączeniu z zależnościami funkcyjnymi, np. jako specyfikację klucza obcego. Jak zobaczymy w rozdziale 11, w tym przypadku nie można liczyć ani na aksjomatyzację, ani na rozstrzygalność implikacji.

Ćwiczenia

1. Dla $\Sigma_1 = \{A \to C, B \to C, CD \to E\}$ pokaż że $\Sigma_1 \models AD \to E$ i $\Sigma_1 \models CDE \to C$.

2. Scharakteryzuj zależności funkcyjne, wielowartościowe i złączeniowe, które są tautologiami.

3. Pokaż, że dla zbiorów dowolnych zależności Σ i Γ zachodzi: (a) $\Sigma \subseteq \Sigma^*$; (b) $(\Sigma^*)^* = \Sigma^*$; (c) jeśli $\Gamma \subseteq \Sigma$, to $\Gamma^* \subseteq \Sigma^*$.

4. Pokaż, że $\Sigma \vdash X \to Y$ i $\Sigma \vdash X \to Z$ implikuje $\Sigma \vdash X \to YZ$ dla dowolnego zbioru zależności funkcyjnych Σ.

5. Zbuduj dowód dla $AB \to F$ z $\{AB \to C, A \to D, CD \to EF\}$ według reguł FD1, FD2, FD3.

6. Pokaż, że następująca reguła jest niepoprawna: jeśli $XW \to Y$ i $XY \to Z$, to $X \to (Z - W)$.

7. Rozważ reguły

   FD4 (pseudoprzechodniość) jeśli $X \to Y$ i $YW \to Z$, to $XW \to Z$;

   FD5 (suma) jeśli $X \to Y$ i $X \to Z$, to $X \to YZ$;

   FD6 (rozkład) jeśli $X \to YZ$, to $X \to Y$.

   Pokaż, że są poprawne. Wskaż dwie reguły spośród FD1–FD6, które tworzą aksjomatyzację dla zależności funkcyjnych.

8. Pokaż, że następująca reguła nie jest poprawna dla zależności wielowartościowych: jeśli $X \twoheadrightarrow Y$ i $Y \twoheadrightarrow Z$, to $X \twoheadrightarrow Z$. Pokaż, że poprawna jest reguła: jeśli $X \twoheadrightarrow Y$ i $Y \twoheadrightarrow Z$, to $X \twoheadrightarrow (Z - Y)$.

10 Zależności zupełne

Class 10

Zależności w ujęciu logicznym

W pełnej ogólności *zależności* definiujemy jako zdania (formuły bez zmiennych wolnych) logiki pierwszego postaci

$$\forall \bar{x}\ \varphi(\bar{x}) \to \exists \bar{y}\ \psi(\bar{x}, \bar{y})\,,$$

gdzie φ, ψ to koniunkcje atomów nad sygnaturą $\mathcal{R} \cup \{=\}$.

Łatwo zauważyć, że można w tej formie zapisać trzy rodzaje zależności opisane w poprzednim rozdziale. Niech $\mathrm{sort}(R) = ABCD$, $\mathrm{sort}(S) = E$.

- R: $AB \to D$ jest równoważne

$$\forall w, x, y, z, y', w'\ R(w, x, y, z) \wedge R(w, x, y', z') \to z = z'\,.$$

- $R\colon \bowtie [AB, CD]$ jest równoważne

$$\forall w, x, y, z, w', x', y', w'\ R(w, x, y, z) \wedge R(w', x', y', z') \to R(w, x, y', z')\,.$$

- $S[E, E] \subseteq R[A, B]$ jest równoważne

$$\forall x\ S(x) \to \exists y, z\ R(x, x, y, z)\,.$$

Zwróćmy uwagę, że pierwsze dwa rodzaje zależności nie wymagają użycia kwantyfikatora egzystencjalnego po prawej stronie implikacji: można je wyrazić za pomocą *zależności zupełnych*, czyli zdań postaci

$$\forall \bar{x}\ \varphi(\bar{x}) \to \psi(\bar{x})\,,$$

gdzie φ, ψ to koniunkcje atomów nad sygnaturą $\mathcal{R} \cup \{=\}$. Ten rodzaj zależności odgrywa szczególną rolę i jemu głównie poświęcony jest niniejszy rozdział.

Łatwo zauważyć, że każdy zbiór zależności Σ jest równoważny zbiorowi Σ' zawierającemu wyłącznie zdania postaci:

- $\forall \bar{x}\ \varphi(\bar{x}) \to x_i = x_j$ (zależności generujące równości),
- $\forall \bar{x}\ \varphi(\bar{x}) \to \exists \bar{y}\ \psi(\bar{x}, \bar{y})$ (zależności generujące krotki),

gdzie φ, ψ to koniunkcja atomów nad $\mathcal{R}$ (bez $=$).

Jeśli Σ zawiera wyłącznie zależności zupełne, to można żądać Σ' zawierającego tylko zależności generujące równości i

- $\forall \bar{x}\ \varphi(\bar{x}) \to R(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$ (zupełne zależności generujące krotki),

gdzie φ to koniunkcja atomów nad $\mathcal{R}$ i $R \in \mathcal{R}$.

Inkluzja zapytań modulo zależności

Obecność więzów sprawia, że problem inkluzji zapytań wymaga przedefiniowania. Jako że teraz rozważamy wyłącznie bazy danych spełniające pewien zestaw więzów, naturalne jest pytanie o inkluzję i równoważność na tych bazach właśnie:

$$\begin{aligned} q_1 \subseteq_\Sigma q_2 &\iff \forall_{\mathbb{A} \models \Sigma}\ q_1^{\mathbb{A}} \subseteq q_2^{\mathbb{A}}\,,\\ q_1 \equiv_\Sigma q_2 &\iff q_1 \subseteq_\Sigma q_2 \text{ oraz } q_2 \subseteq_\Sigma q_1\,. \end{aligned}$$

W następnej części rozdziału rozwiniemy narzędzie, które w końcu pozwoli na rozstrzyganie inkluzji modulo zbiór zależności zupełnych.

Pogoń

Do rozstrzygania inkluzji modulo zależności posłuży nam procedura zwana *pogonią* (ang. *chase*). Polega ona na nasyceniu zapytań koniunkcyjnych, tak aby były jak najbardziej selektywne, ale pozostały równoważne wyjściowym (modulo Σ). Dokładniej, będziemy poszerzać zapytanie koniunkcyjne q o wszystkie

dodatkowe atomy, które i tak wynikają z q na mocy zależności. W tym rozdziale rozważamy jedynie zależności zupełne.

Niech σ będzie zależnością zupełną $\forall \bar{x}\ \varphi(\bar{x}) \rightarrow A(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$, gdzie $A \in \mathcal{R} \cup \{=\}$. Zastosowanie zależności σ do zapytania koniunkcyjnego $q\colon Q \leftarrow Body$ daje w wyniku zapytanie $q'\colon Q' \leftarrow Body'$ otrzymane jak następuje:

1. Weź podstawienie $\theta : \bar{x} \rightarrow \mathrm{var}(q) \cup \mathrm{const}(q)$ takie, że $\varphi(\theta(\bar{x}))$ jest zawarte w $Body$.
2. Jeśli $A(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) = R(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$, niech $Q' = Q$ i
$$Body' = Body \cup \{A(\theta(x_{i_1}), \theta(x_{i_2}), \ldots, \theta(x_{i_k}))\}\,.$$
3. Jeśli $A(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$ jest równością $x_i = x_j$, to $Body', Q'$ są otrzymane przez zastąpienie wszystkich wystąpień $\theta(x_i)$ przez $\theta(x_j)$, o ile $\theta(x_i)$ jest zmienna. Jeśli $\theta(x_i)$ jest stałą, a $\theta(x_j)$ jest zmienna, to postępujemy symetrycznie. Jeśli $\theta(x_i)$ i $\theta(x_j)$ są tą samą stałą, to $q' = q$, a jeśli są dwiema różnymi stałymi, to q' jest zapytaniem pustym $q^{\emptyset}$.

Dowód poniższego faktu jest łatwym ćwiczeniem.

Fakt 3. *Jeśli Σ jest zbiorem zupełnych zależności generujących krotki i zależności generujących równości oraz q' jest otrzymane z q przez zastosowanie $\sigma \in \Sigma$, to $q' \equiv_\Sigma q$.*

Ustalmy Σ i q_0. Rozważmy maksymalny (skończony lub nieskończony) ciąg zapytań koniunkcyjnych $q_0, q_1, q_2, q_3, \ldots$, w którym q_{i+1} jest otrzymane z q_i poprzez zastosowanie zależności z Σ oraz $q_{i+1} \neq q_i$. Zauważmy, że ciąg taki musi być skończony, gdyż nie dokładamy nowych zmiennych do zapytania. Ostanie zapytanie q_N jest *wynikiem* pogoni Σ za q_0.

A priori wynik pogoni Σ za q_0 nie jest jednoznacznie zdefiniowany. Można pokazać, że przy sensownym wybieraniu nazwy dla sklejonych zmiennych, wynik ten nie zalezy od wyborów zależności i θ w kolejnych krokach [1, Twierdzenie 8.4.18]. Dla naszych potrzeb jednak nie będzie to miało znaczenia. Każdy wynik będzie dla nas jednakowo dobry. Nieco nieformalnie będziemy pisali $\mathrm{chase}(\mathrm{q}_0, \Sigma)$, mając na myśli dowolnie wybrany wynik.

Fakt 4. *Niech Σ będzie zbiorem zupełnych zależności i niech q będzie zapytaniem koniunkcyjnym. Wtedy*

1. $\mathrm{chase}(q, \Sigma) \equiv_\Sigma q$*;*
2. $\mathrm{chase}(q, \Sigma)$ *interpretowane jako struktura spełnia* Σ*.*

Dowód. Pierwszy punkt wynika z definicji i z Faktu 3. Drugi punkt: gdyby jakaś zależność nie była spełniona, to można by wydłużyć pogoń, co przeczy maksymalności w definicji $\mathrm{chase}(q, \Sigma)$. □

Weźmy zapytania q, q'. Na mocy Faktu 4 mamy $\mathrm{chase}(q, \Sigma) \equiv_\Sigma q$ oraz $\mathrm{chase}(q', \Sigma) \equiv_\Sigma q'$, czyli

$$q \equiv_\Sigma q' \iff \mathrm{chase}(q, \Sigma) \equiv_\Sigma \mathrm{chase}(q', \Sigma)\,.$$

Zatem niczego nie popsuliśmy, ale nie widać jeszcze, dlaczego poprawiliśmy. Odpowiedź daje poniższe twierdzenie.

Twierdzenie 3. *Niech Σ to zbiór zupełnych zależności, a q, q' to zapytania koniunkcyjne.*

1. $q \subseteq_\Sigma q' \iff \mathrm{chase}(q, \Sigma) \subseteq \mathrm{chase}(q', \Sigma)$.
2. $q \equiv_\Sigma q' \iff \mathrm{chase}(q, \Sigma) \equiv \mathrm{chase}(q', \Sigma)$.

Punkt drugi twierdzenia wynika wprost z pierwszego. Dowód pierwszego punktu polega na zastosowaniu poniższego lematu dla $\mathcal{F} = \{\mathbb{A} \mid \mathbb{A} \models \Sigma\}$.

W lemacie $q_1 \subseteq_\mathcal{F} q_2$ oznacza $\forall_{\mathbb{A}\in\mathcal{F}}\ q_1^\mathbb{A} \subseteq q_2^\mathbb{A}$. Podobnie definiujemy $\equiv_\mathcal{F}$. Korzystamy również z wprowadzonej w rozdziale 5 (strona **??**) interpretacji zapytań koniunkcyjnych jako baz danych.

Lemat 1. *Niech $\mathcal{F}$ będzie rodziną baz danych zamkniętą na izomorfizmy. Niech $q_1, q_2, q_1', q_2' \in \mathrm{CQ}$ spełniają $q_i \equiv_\mathcal{F} q_i'$ oraz $q_i' \in \mathcal{F}$ (traktowane jako baza danych) dla $i = 1, 2$. Wtedy*

$$q_1 \subseteq_\mathcal{F} q_2 \iff q_1' \subseteq q_2'\,.$$

Dowód. Implikacja $\Leftarrow$ jest oczywista. Aby pokazać $\Rightarrow$ załóżmy że $q_1 \subseteq_\mathcal{F} q_2$. Wystarczy pokazać, że istnieje homomorfizm $q_2' \to q_1'$. Niech $\mathbb{A}_{q_1'}$ będzie bazą danych odpowiadającą zapytaniu q_1'. Na mocy założeń lematu mamy

$$q_1'(\mathbb{A}_{q_1'}) \subseteq q_1(\mathbb{A}_{q_1'}) \subseteq q_2(\mathbb{A}_{q_1'}) \subseteq q_2'(\mathbb{A}_{q_1'})\,.$$

Pamiętamy, że $\mathrm{ans}_{q_1'} \in q_1'(\mathbb{A}_{q_1'})$, gdzie $\mathrm{ans}_{q_1'}$ to krotka odpowiadająca nagłówkowi zapytania q_1'. Zatem mamy $\mathrm{ans}_{q_1'} \in q_2'(\mathbb{A}_{q_1'})$, a to oznacza, że istnieje homomorfizm $q_2' \to q_1'$. □

Rozstrzyganie inkluzji i implikacji

Procedura pogoni i Twierdzenie 3 pozwalają na wyciągnięcie dwóch istotnych wniosków.

Wniosek 1. *Dla danych $q, q' \in \mathrm{CQ}$ i zbioru zupełnych zależności Σ można rozstrzygać $q \subseteq_\Sigma q'$ w EXPTIME.*

Dowód. Rozmiar $\mathrm{chase}(q, \Sigma)$ jest pojedyńczo wykładniczy zwn Σ i q, zatem pogoń będzie miała długość pojedyńczo wykładniczą. Każdy krok pogoni (aplikacja zależności) da się zrobić w czasie niedeterministycznym wielomianowym: wystarczy zgadnąć zależność σ i wartościowanie θ (wartościowanie ma rozmiar wielomianowy zwn na σ).

Można zatem obliczyć $\mathrm{chase}(q, \Sigma)$ i $\mathrm{chase}(q', \Sigma)$ w EXPTIME. Dalej należy znaleźć homomorfizm z $\mathrm{chase}(q', \Sigma)$ w $\mathrm{chase}(q, \Sigma)$. Wprawdzie oba zapytania mają rozmiar wykładniczy, ale $\mathrm{chase}(q', \Sigma)$ ma tylko wielomianowo wiele zmiennych. Można zatem sprawdzić wszystkie kandydatury na homomorfizm: jest ich pojedyńczo wykładniczo dużo. Sprawdzenie pojedynczej kandydatury wymaga czasu wykładniczego, bo rozmiar zapytań jest wykładniczy. □

Wniosek 2. *Dla zadanego zbioru zależności zupełnych Σ i pojedynczej zależności (nie koniecznie zupełnej) σ, można rozstrzygnąć w EXPTIME, czy $\Sigma \models \sigma$.*

Dowód. Niech σ będzie postaci $\forall \bar{x}\ \varphi(\bar{x}) \implies \exists \bar{y}\ \psi(\bar{x}, \bar{y})$. Wystarczy zauważyć, że $\Sigma \models \sigma$ jest równoważne $\{\bar{x} \mid \varphi(\bar{x})\} \subseteq_\Sigma \{\bar{x} \mid \exists \bar{y}\ \psi(\bar{x}, \bar{y}) \wedge \varphi(\bar{x})\}$. □

Ćwiczenia

1. Wykaż, że
   (a) każdy zbiór zależności jest równoważny zbiorowi zależności generujących krotki i zależności generujących równości;
   (b) każdy zbiór zupełnych zależności jest równoważny zbiorowi zależności generujących równości i zupełnych zależności generujących krotki.
2. Pokaż, że $\text{chase}(q, \Sigma)$ może mieć rozmiar wykładniczy zwn rozmiar Σ i q.
3. Pokaż, że dla dowolnych $q, q' \in \text{CQ}$ i dowolnego zbioru zależności zupełnych Σ zachodzi $q \subseteq_\Sigma q' \iff \text{chase}(q, \Sigma) \subseteq q'$.
4. Pokaż, że dla zbioru zależności zupełnych σ i dowolnej zależności zupełnej σ, $\Sigma \models \sigma$ nad strukturami skończonymi (ozn. $\Sigma \models_{\text{fin}} \sigma$) wtedy i tylko wtedy gdy $\Sigma \models \sigma$ nad dowolnymi strukturami (ozn. $\Sigma \models_{\text{unr}} \sigma$).

   Wskazówka: jeśli formuła typu $\exists^*\forall^*$ ma model, to ma (mały) model skończony.

   Zwróć uwagę, że można w ten sposób rozstrzygać implikację dla zupełnych zależności w czasie NEXPTIME.
5. Pokaż, że dla dowolnych zależności powyższa równoważność nie zachodzi. Wskazówka: $\Sigma = \{A \to B,\ R[A] \subseteq R[B]\}$, $\sigma = R[B] \subseteq R[A]$.

11 Zależności – przypadek ogólny

Class 11

Nierozstrzygalność

W poprzednim rozdziale poznaliśmy metodę pogoni pozwalającą na rozstrzyganie inkluzji zapytań koniunkcyjnych modulo zależności zupełne i implikacji zależności zupełnych. Dla zależności niezupełnych implikacja i inkluzja jest nierozstrzygalna.

Przypomnijmy najpierw, że dla niezupełnych zależności implikacja w modelach skończonych $\models_{\text{fin}}$ i implikacja w dowolnych modelach $\models_{\text{unr}}$ nie są tożsame. W obu przypadkach problem jest nierozstrzygalny.

Theorem 17.

1. *Problem ustalenia, czy dla zadanego zbioru niezupełnych zależności Σ i zadanej niezupełnej zależności σ zachodzi $\Sigma \models_{\text{fin}} \sigma$, jest nierozstrzygalny.*

2. *Problem ustalenia, czy dla zadanego zbioru niezupełnych zależności Σ i zadanej niezupełnej zależności σ zachodzi $\Sigma \models_{\text{unr}} \sigma$, jest nierozstrzygalny.*

Dowód. Redukcja problemu słów dla monoidów do problemu implikacji dla zależności funkcyjnych i zależności inkluzji. Dowód przebiega identycznie dla obu punktów, z tym że w pierwszym punkcie korzystamy z nierozstrzygalności problemu słów dla monoidów skończonych, a w drugim punkcie z nierozstrzygalności problemu słów dla dowolnych monoidów. Szczegóły można znaleźć na stronach 199–202 w [1]. □

Wniosek 3. *Problem ustalenia, czy dla danych $q, q' \in \mathrm{CQ}$ i dla danego zbioru zależności niezupełnych Σ zachodzi $q \subseteq_\Sigma q'$, jest nierozstrzygalny.*

Dowód. Podobnie jak w dowodzie Wniosku 2 pokazujemy, że implikacja redukuje się do inkluzji zapytań koniunkcyjnych i korzystamy z pierwszego punktu twierdzenia powyżej. □

Brak aksjomatyzacji

Aksjomatyzacją dla klasy więzów $\mathcal{W}$ nazwiemy zbiór reguł (ustalonych) $\mathcal{A}$ postaci „jeśli $\sigma_1, \sigma_2, \ldots, \sigma_n$, to σ", który jest pełny i poprawny (tzn. dla dowolnych $\Sigma \subseteq \mathcal{W}, \sigma \in \mathcal{W}$ zachodzi $\Sigma \models_{\text{fin}} \sigma \iff \Sigma \vdash_{\mathcal{A}} \sigma$).

Aksjomatyzacja jest rekurencyjnie przeliczalna, gdy istnieje algorytm, który wypisuje wszystkie reguły $\mathcal{A}$ w pewnej kolejności (jest to założenie istotnie słabsze, niż rekurencyjność).

Wniosek 4. *Dla klasy zależności niezupełnych nie istnieje aksjomatyzacja rekurencyjnie przeliczalna.*

Dowód. Przypuśćmy, że istnieje. Pokażemy, że wtedy problem implikacji jest rozstrzygalny. Dla zadanych Σ, σ robimy jednocześnie dwie rzeczy. Po pierwsze, korzystając z algorytmu wypisującego reguły $\mathcal{A}$, generujemy wszystkie dowody z Σ i patrzymy czy przypadkiem ostatnia zależność to nie jest σ. Po drugie, testujemy po kolei wszystkie skończone bazy danych, szukając takiej, która spełnia Σ ale nie spełnia σ. Któryś z tych dwóch wątków naszego obliczenia zakończy się sukcesem: jeśli implikacja zachodzi, to znajdziemy dowód. Jeśli nie zachodzi, to znajdziemy kontrprzykład.

Tak więc przypuszczenie, że istnieje rekurencyjnie przeliczalna aksjomatyzacja, prowadzi do sprzeczności z Twierdzeniem 17. □

Powyższa metoda pokazuje, że nie może istnieć aksjomatyzacja implikacji (nad skończonymi strukturami) dla klas więzów, dla których implikacja jest nierozstrzygalna. W szczególności zatem, klasa zależności funkcyjnych i zależności inkluzyjnych nie posiada aksjomatyzacji rekurencyjnie przeliczalnej.

Metoda ta nie ma jednak zastosowania do klas więzów w których implikacja jest rozstrzygalna. Innym sposobem powiedzenia, że aksjomatyzacja jest porządna jest narzucenie warunku na ilość zależności ujętych w pojedynczych regułach.

Aksjomatyzację nazwiemy *k-arną*, jeśli każda jej reguła jest postaci „jeśli $\sigma_1, \sigma_2, \ldots, \sigma_\ell$, to σ" dla $\ell \leq k$.

Fakt 5. *Nie istnieje k-arna aksjomatyzacja dla klasy zależności funkcyjnych i inkluzyjnych.*

Dowód. Strony 204–206 w [1]. □

Nieskończona pogoń

Jak już wiemy, w przypadku niezupełnych zależności pogoń nie da nam rozstrzygalności implikacji. Mimo to, procedura ta może być przydatna.

W przypadku niezupełnych zależności generujących krotki, zastosowanie $\forall \bar{x}\ \varphi(\bar{x}) \to \exists \bar{y}\ \psi(\bar{x}, \bar{y})$ do q przebiega następująco:

1. Weź wszystkie podstawienia $\theta\colon \bar{x} \to \mathrm{var}(q) \cup \mathrm{const}(q)$, takie że każdy atom $\varphi(\theta(\bar{x}))$ występuje w ciele q, dla których nie istnieje $\theta'\colon \bar{y} \to \mathrm{var}(q) \cup \mathrm{const}(q)$, takie że każdy atom $\psi(\theta(\bar{x}), \theta'(\bar{y}))$ występuje w ciele q.

2. Ustal dla każdego θ podstawienie θ' przyporządkowujące każdej zmiennej z $\bar{y}$ świeżą zmienna (inną dla każdego θ).

3. Dodaj do ciała zapytania q wszystkie atomy występujące w $\psi(\theta(\bar{x}), \theta'(\bar{y}))$.

Zależności generujące równości stosujemy tak, jak dawniej. Podczas obliczania pogoni stosujemy dwie reguły:

- Jeśli zaczniesz aplikować zależności generujące równości, to rób to tak długo jak się da.

- Jeśli w pewnym momencie pewna zależność daje się zastosować, to musisz zagwarantować, że w końcu zostanie zastosowana.

Intuicyjnie, przetrzeganie tych reguł gwarantuje, że wszystkie zależności zostaną wzięte pod uwagę.

Przy tak zdefiniowanej procedurze pogoni, wygenerowany ciąg zapytań może być nieskończony. Nie jest zatem zupełnie oczywiste, co powinno być wynikiem takiej pogoni. Na przykład wzięcie wszystkich atomów ze wszystkich zapytań nie jest dobrym pomysłem, bo dowolnie późno w ciągu mogą wystąpić sklejenia zmiennych. Rozwiązanie polega na wzięciu tych atomów, które zostają na zawsze:

$$\mathrm{chase}(q, \Sigma) = \{A \mid \exists_n \forall_{m>n}\ A \in q_n\}$$

gdzie $q_1, q_2, \ldots$ jest pewnym przebiegiem procedury pogoni.

Zauważmy, że wynik pogoni nie jest zapytaniem koniunkcyjnym, bo może być nieskończony. Można natomiast na niego spojrzeć jak na strukturę, która musi się przekształcać homomorficznie na każdą bazę danych, która spełnia jednocześnie q i Σ. Łatwo sprawdzić, że:

- $\mathrm{chase}(q, \Sigma) \models \Sigma$

- $q \subseteq^{\mathrm{unr}}_{\Sigma} q'$ wtedy i tylko wtedy, gdy istnieje homomorfizm $q' \to \mathrm{chase}(q, \Sigma)$.

Uwaga: unr powyżej oznacza, że rozważamy inkluzję nad wszystkimi modelami spełniającymi Σ: skończonymi i nie skończonymi. Aby to lepiej zrozumieć, załóżmy, że q, q' są zapytaniami boole'owskimi. Jeśli q' jest spełnione w $\text{chase}(q, \Sigma)$, to musi być spełnione w każdym modelu $\Sigma \cup \{q\}$. Jeśli nie jest spełnione w $\text{chase}(q, \Sigma)$, to znaleźliśmy nieskończony model $\Sigma \cup \{q\}$ w którym q' nie jest spełnione. Czyli nie zachodzi $q \subseteq_\Sigma^{\text{unr}} q'$. Nic to jednak nam nie mówi na temat tego, czy wszystkie skończone modele $\Sigma \cup \{q\}$ spełniają q'.

Ćwiczenia

1. Pokaż przykład zapytania q i zbioru zależności Σ, dla których istnieje nieskończona pogoń i skończona pogoń.

12 Niepełne dane

Class 12

Wykład w miarę precyzyjnie podążał za (krótkim i czytelnym) rozdziałem 19 w [1], z pominięciem punktów *Languages with Recursion* i *Dependencies* z sekcji 19.3 oraz całej sekcji 19.5.

13 Wymiana danych

Class 13

Wykład w całości oparty o slajdy *Data Exchange and Schema Mappings* dostępne na stronie przedmiotu.

Ćwiczenia

1. Pokaż, że dla przekształcenia zadanego przez zależności generujące krotki i zależności generujące równości, roziązanie kanoniczne jest uniwersalne.

2. Pokaż, że dla ustalonego $\mathcal{M}$ zadanego przez zależności generujące krotki, rozstrzygnięcie, czy $(I, J) \in \mathcal{M}$ jest w PTIME.

3. Pokaż, że dla ustalonych $\mathcal{M}_1$ i $\mathcal{M}_2$ zadanych przez zależności generujące krotki rostrzygnięcie, czy $(I, J) \in \mathcal{M}_1 \circ \mathcal{M}_2$ jest NP-zupełne.

4. Pokaż, że nierozstrzygalne jest $\text{certain}_{\mathcal{M}}(Q, I) = \text{true}$ dla $\mathcal{M}$ zadanego przez zależności generujące krotki i $Q \in \text{RA}$.

5. *Rdzeniem* struktury skończonej $\mathbb{A}$ nazywamy minimalną podstrukturę $\mathbb{A}_0$, taką żee istnieje homomorfizm $h \colon \mathbb{A} \to \mathbb{A}_0$. Pokaż, że każda struktura skończona ma dokładnie jeden rdzeń (z dokładnością do izomorfizmu).

6. Pokaż, że problem sprawdzenia, czy graf G jest rdzeniem grafu G' jest NP-trudny i że da się go rozwiązać wykonując pewną procedurę NP i pewną porocedurę co-NP. (Ta klasa założoności nazywa się DP.)

14 Zewnętrzna charakteryzacja klas przekształceń schematów

Wykład oparty o artykuł Baldera ten Cate i Phokiona Kolaitisa [2], dostępny na stronie pierwszego autora.

Dotychczas pracowalismy z przekształceniami schematów opisanymi przez zbiory zależności generujących krotki, tj. zależności postaci $\forall \bar{x}\, \varphi(\bar{x}) \to \exists \bar{y}\, \psi(\bar{x}, \bar{y})$. Pokazaliśmy, że maja one różne dobre własności i ogólnie nadają się do wymiany danych. Nie pytalismy jednak dlaczego zajmujemy się akurat takimi przekształceniami. Teraz zmienimy perspektywę i spróbujemy zrozumieć tę klasę przekształceń w oderwaniu od jej syntaktycznej reprezentacji. Podamy pewien nieduży zestaw własności, który definiuje dokładnie te przekształcenia, które można zdefiniować za pomocą skończonego zbioru zależności generujących krotki.

W podejściu, które teraz przyjmujemy, przekształcenie schematów to po prostu relacja dwuargumentowa łącząca instancje źródłowego schematu $\mathcal{R}_s$ z odpowiadającymi im instancjami decelowego schematu $\mathcal{R}_s$, czyli podzbiór

$$\mathcal{M} \subseteq \text{Instancje}(\mathcal{R}_s) \times \text{Instancje}(\mathcal{R}_t)\,.$$

Zamknięcie na homomorfizmy celu. Przekształcenie $\mathcal{M}$ jest zamknięte na homomorfizmy celu jeśli dla dowolnych $(I, J) \in \mathcal{M}$ i dla dowolnego homomorfizmu $h\colon J \to J'$ stałego na $\text{adom}(I)$ zachodzi $(I, J) \in \mathcal{M}$.

Gwarancja uniwersalnych rozwiązań. Przekształcenie $\mathcal{M}$ gwarantuje uniwersalne rozwiązania, jeśli dla każdego I istnieje uniwersalne rozwiązanie J względem $\mathcal{M}$, tzn. takie J że $(I, J) \in \mathcal{M}$ oraz jeśli $(I, J') \in \mathcal{M}$, to istnieje homomorfizm $h : J \to J'$ stały na $\text{adom}(I)$.

Odzwierciedlanie homomorfizmów źródła. Przekształcenie $\mathcal{M}$ odzwierciedla homomorfizmy źródła, jeśli dla dowolnych I, I', J, J', takich że J jest rozwiązaniem uniwersalnym dla I, a J' jest jakimś rozwiązaniem dla I', każdy homomorfizm $h\colon I \to I'$ rozszerza się do homomorfizmu $\tilde{h}\colon J \to J'$.

Modularność. Przekształcenie $\mathcal{M}$ jest modularne, jeśli istnieje taka stała n zależna tylko od $\mathcal{M}$, że dla każdej pary $(I, J) \notin \mathcal{M}$ istnieje $I' \subseteq I$ taka że $\text{adom}\, I' \leq n$ oraz $(I', J) \notin \mathcal{M}$.

Twierdzenie 4. *Przekształcenie schematów $\mathcal{M}$ jest definiowalne za pomocą skończonego zbioru zależności generujących krotki wtedy i tylko wtedy gdy (a) jest zamknięte na homomorfizmy celu, (b) gwarantuje rozwiązania uniwersalne, (c) odzwierciedla homomorfizmy źródła oraz (d) jest modularne.*

Dowód. Załóżmy, że $\mathcal{M}$ jest dane przez skończony zbiór zależności generujących krotki. Zamkniętość na homomorfizmy celu bierze się stąd, że złożenie homomorfizmu z waluacją zmiennych spełeniającą daną zależności w J daje waluacje spełniającą tę zależność w J'. Fakt, że każda instancja źródłowa ma rozwiązanie uniwersalne pokazalismy na poprzednim wykładzie (rozwiązanie kanoniczne, konstruowane przez algorytm pogoni, jest uniwersalne).

Używając podobnej metody pokażemy, że $\mathcal{M}$ odzwierciedla homomorfizmy źródła. Weźmy I, I', J, J', takie że J jest rozwiązaniem uniwersalnym dla I, a J' jest jakimś rozwiązaniem dla I' i niech $h\colon I \to I'$ będzie homomorfizmem. Załóżmy na początek, że J jest rozwiązaniem kanonicznym. Zdefiniujemy $\tilde{h}\colon J \to J'$ indukcyjnie według konstrukcji roziązania kanonicznego. Aplikacja zależności $\forall \bar{x}\, \varphi(\bar{x}) \to \exists \bar{y}\, \psi(\bar{x}, \bar{y})$ dla krotki $\bar{a}$ polega na dodaniu do rozwiązania J nowych wartości (nuli) $\bar{b}$ o relacjach opisanych przez $\psi(\bar{a}, \bar{b})$. Jak zadać $\tilde{h}$ na tych wartościach? Skoro h jest homomorfizmem, to $I' \models \varphi(h(\bar{a}))$. Zatem $J' \models \psi(h(\bar{a}), \bar{c})$ dla pewnej krotki $\bar{c}$, bo J' spełnia zależności. Wystarczy położyć $\tilde{h}(b_i) = c_i$. Otrzymamy w ten sposób homomorfizm z obecnego przybliżenia rozwiązania J. Dokładając dalsze elementy do J nie zepsujemy go, bo nigdy nie dodamy żadnych nowych krotek zawierających nule dodane w poprzednim kroku. Zatem na końcu otrzymamy homomorfizm $\tilde{h}\colon J \to J'$ rozszerzający h.

Jeśli J nie jest rozwiązaniem kanoniczym dla I, to niech K nim będzie. J jest rozwiązaniem uniweralnym, więc istnieje homomorfizm $g\colon J \to K$ stały na $\operatorname{adom} I$. Z przypadku szcególnego wiemy, że istnieje homomorfizm $\tilde{h}\colon K \to J'$ rozszerzający h. Złożenie $g \circ \tilde{h}$ daje homomorfizm z J w J' rozszerzający h.

Pozostaje pokazać modularność. Niech n będzie maksymalną liczbą zmiennych użytych po lewej stronie zależności. Weźmy $(I, J) \notin \mathcal{M}$. W takim razie istnieje zależność $\forall \bar{x}\, \varphi(\bar{x}) \to \exists \bar{y}\, \psi(\bar{x}, \bar{y})$ i krotka $\bar{a}$, taka że $I \models \varphi(\bar{a})$, ale J nie spełnia $\psi(\bar{a}, \bar{b})$ dla żadnej krotki $\bar{b}$. Niech I' będzie obcięciem I do wartości użytych w $\bar{a}$ (jest ich nie więcej niż n). Oczywiście $I' \models \varphi(\bar{a})$, zatem $(I', J) \notin \mathcal{M}$. Zatem $\mathcal{M}$ jest modularne.

Teraz udowodnimy, że zachodzi implikacja w drugą stronę. Weźmy przekształcenie spełniające warunki (a)–(d). Niech n będzie stałą modularności $\mathcal{M}$. Ustalmy zbiór n stałych, $A = \{a_1, a_2, \ldots, a_n\}$. Niech $I_1, I_2, \ldots, I_m$ będą wszystkimi instancjami o dziedzinie aktywnej zawartej w A. Niech J_i będzie rozwiązaniem uniweralnym dla I_i. Niech $\varphi_i(\bar{x})$ będzie koniunkcją wszystkich krotek z I_i, w której każdą stałą a zamieniono na zmienną x_a. Podobnie definiujemy $\psi_i(\bar{x}, \bar{y})$, zamieniając każego nula na nową zmienną. Niech σ_i będzie zależnością $\forall \bar{x}\, \varphi_i(\bar{x}) \to \exists \bar{y}\, \psi_i(\bar{x}, \bar{y})$.

Udowodnimy, że $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ definiuje $\mathcal{M}$. Weźmy najpierw $(I, J) \in \mathcal{M}$. Przypuśćmy, że $I \models \varphi_i(\bar{c})$. Zatem istnieje homomorfizm $h\colon I_i \to I$, spełniający $h(a) = c_a$, gdzie c_a jest wartością zmiennej x_a daną przez wartościowanie $\bar{c}$. Na mocy odzwierciedlania homomorfizmów źródła, istnieje $\tilde{h}\colon J_i \to J$, rozszerzający h. Ten homomorfizm zadaje wartościowanie zmiennych $\bar{y}$, $\tilde{h}(\bar{y})$, takie że $J \models \psi(\bar{c}, \tilde{h}(\bar{y}))$.

W drugą stronę, weźmy (I, J) spełniające Σ i załóżmy, że $(I, J) \notin \mathcal{M}$. Na mocy (d), istnieje $I' \subseteq I$, takie że $\operatorname{adom} I' \leq n$ oraz $(I', J) \notin \mathcal{M}$. Oczywiście (I', J) spełnia zależności Σ. Z konstrukcji wynika, że istnieje I_i takie że $I' \simeq I_i$. Niech $\eta : I' \to I_i$ będzie izomorfizmem. Mamy zatem $I' \models \varphi_i(\bar{c})$, gdzie $\bar{c}$ jest wartościowaniem zadanym przez η^{-1}, tzn. $c_k = \eta^{-1}(a_{j_k})$, gdzie $\operatorname{adom} I_i = \{a_{j_1}, a_{j_2}, \ldots, a_{j_\ell}\}$. Na mocy (c), istnieje $\tilde{\eta}\colon J' \to J_i$ rozszerzający η, gdzie J' jest dowolnym rozwiązaniem uniwersalnym dla I', istniejącym na mocy (b). Skoro (I', J) spełnia Σ, to $J \models \psi_i(\bar{c}, \bar{d})$, dla pewnej krotki $\bar{d}$. Zatem istnieje homomorfizm $g\colon J_i \to J$, taki że $g(a_{j_k}) = \bar{c}_k = \eta^{-1}(a_{j_k})$. Złożenie $\tilde{\eta} \circ g$ daje

homomorfizm z J' w J, stały na $\operatorname{adom} I'$. Na mocy (a), J jest rozwiązaniem dla I', co prowadzi do sprzeczności. □

Podobny wynik można uzyskać, dla zupełnych zależności generujących krotki, potrzebny jest tylko jeden dodatkowy warunek.

Zamknięcie na przecięcia celu. Przekształcenie $\mathcal{M}$ jest zamknięte na przecięcia celu, jeśli dla każdych instancji I, J_1, J_2,

$$(I, J_1), (I, J_2) \in \mathcal{M} \implies (I, J_1 \cap J_2) \in \mathcal{M}\,.$$

Twierdzenie 5. *Przekształcenie schematów $\mathcal{M}$ jest definiowalne za pomocą skończonego zbioru zupełnych zależności generujących krotki wtedy i tylko wtedy gdy (a) jest zamknięte na homomorfizmy celu, (b) gwarantuje rozwiązania uniwersalne, (c) odzwierciedla homomorfizmy źródła, (d) jest modularne oraz (e) jest zamknięte na przecięcia celu.*

Dowód. Załóżmy, że $\mathcal{M}$ jest zadane przez skończony zbiór zupełnych zależności generujących krotki. Zauważmy, że rozwiązanie kanoniczne dla takiego przekształcenia nie zawiera nuli, a zatem każde inne rozwiązanie musi być nadzbiorem rozwiązania kanonicznego. Stąd mamy zamknięcie na przecięcia celu. Pozostałe własności udowodniliśmy w Twierdzeniu 4.

Przejdźmy do drugiej implikacji. Pokażemy, że przy założeniach (a)–(e) każda instancja I ma rozwiązanie uniwersalne J, takie że $\operatorname{adom} J \subseteq \operatorname{adom} I$. W tym celu weźmy dowolne rozwiązanie uniwersalne J_1 dla I. Niech J_2 będzie jego izomorficzną kopią, zgodną z J_1 na $\operatorname{adom} I$, ale rozłączną poza $\operatorname{adom} I$. Skoro J_2 jest izomorficzne z J_1, to J_2 również jest rozwiązaniem dla I. Na mocy (e), $J = J_1 \cap J_2$ jest rozwiązaniem dla I. Z konstrukcji wynika, że $\operatorname{adom} J \subseteq \operatorname{adom} I$. J jest uniwersalne, bo inkluzja $J \subseteq J_1$ jest homomorfizmem w rozwiązanie uniwersalne, a zatem dla każdego rozwiązania J' istnieje homomorfizm z J w J' dany przez złożenie tej inkluzji z $h\colon J_1 \to J'$ gwarantowanym przez uniwersalność J_1.

Teraz możemy powtórzyć niemal dosłownie argument z Twierdzenia 4, z tą różnicą, że rozwiązanie uniwersalne J_i będzie zawierało wyłącznie stałe z I_i, a zatem odpowiadające mu zależności będą zupełne. □

Aby zobaczyć, że nie można się pozbyć warunku (d), rozważmy przekształcenie zadane przez nieskończony zbiór zależności

$$\left\{\forall x_1, x_2, \ldots, x_n\, E(x_1, x_2) \wedge E(x_2, x_3) \wedge \cdots \wedge E(x_{n-1}, x_n) \to T(x_1, x_n) \;\middle|\; n > 0\right\}\,.$$

Łatwo sprawdzić, że to przekształcenie spełnia warunki (a),(b),(c) oraz (e), ale oczywiście nie jest modularne. Można pokazać, że nie jest ono definiowalne przez skończony zbiór zależności generujących krotki.

Można jednak udowodnić, że jeśli przekształcenie $\mathcal{M}$ jest zadane przez zdanie logiki pierwszego rzędu, to zakładanie modularności jest niepotrzebne.

Twierdzenie 6. *Przekształcenie schematów $\mathcal{M}$ zadane zdaniem logiki pierwszego rzędu jest definiowalne za pomocą skończonego zbioru zupełnych zależności generujących krotki wtedy i tylko wtedy gdy jest zamknięte na homomorfizmy celu, gwarantuje rozwiązania uniwersalne, odzwierciedla homomorfizmy źródła, oraz jest zamknięte na przecięcia celu.*

Ćwiczenia

1. Pokaż, że dowolne przekształcenie $\mathcal{M}$ zamknięte na homomorfizmy celu i na przecięcia celu spełnia następujący warunek:

$$(I, J) \in \mathcal{M} \iff (I, J \restriction_{\mathrm{adom}\, I}) \in \mathcal{M}\,.$$

2. Pokaż, że dla $\mathcal{M}$ zadanego zdaniem logiki pierwszego rzędu, nierozstrzygalne jest, czy $\mathcal{M}$ jest definiowalne za pomocą (zupełnych) zależności generujących krotki.

   Wskazówka: Wykorzystaj nierozstrzygalność spełnialności formuł FO w modelach skończonych oraz zamknięcie na homomorfizmy celu przekształceń definiowalnych za pomocą zależności generujących krotki.

   Rozwiązanie: $\{\varphi \to \neg\exists x\, R(x)\}$ jest definiowalne wtw, gdy φ jest niespełnialne.

3. Pokaż, że inkluzja przekształceń zadanych przez zależności generujące krotki jest z NP-zupełna.

   Wskazówka: Rozważ rozwiązanie kanoniczne dla instancji reprezentującej poprzednika zależności.

4. Niech $\mathcal{M}$ będzie zadane przez zależności generujące krotki. Niech $\mathcal{M}'$ powstaje z $\mathcal{M}$ poprzez usunięcie kwantyfikatorów egzystencjalnych i wykreślenie wszystkich atomów zawierających zmienne kwantyfikowane egzsystencjalnie. Pokaż, że $\mathcal{M}$ jest definiowalne przez zupełne zależności wtw, gdy $\mathcal{M} = \mathcal{M}'$.

5. Pokaż, że definiowalność za pomocą zupełnych zależności przekształcenia $\mathcal{M}$ zadanego przez dowolne zależności generujące krotki jest NP-zupełna.

References

[1] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.

[2] B. ten Cate, Ph. Kolaitis. Structural characterizations of schema-mapping languages. Proc. ICDT 2009.

`http://users.soe.ucsc.edu/~btencate/papers/icdt09.pdf`