A Primer on the Containment Problem for Conjunctive Queries

Jef Wijsen

February 1, 2021

Abstract

The aim of this primer is to provide a step by step introduction to the theory of conjunctive queries. Conjunctive queries are very common in database systems, and constitute the building blocks for more complicated queries. They correspond to the select-project-Cartesian product queries in relational algebra (commonly called SPC queries[1]), and are easily encoded as SELECT-FROM-WHERE queries in SQL. The theory of conjunctive queries has been developed in the database community since the early days of the relational database model, and turns out to have close connections to other domains of computer science. In this primer, we show some results in this theory, which are at the same time fundamental, elegant, and easily understandable to students familiar with the relational database model.

This primer assumes that students have already studied *A Datalog Primer*, which is available at `http://informatique.umons.ac.be/ssi/teaching/bdIImons/primerDatalog.pdf`. Suggestions, corrections, and comments are very welcome at `jef.wijsen@umons.ac.be`.

Contents

[1] These correspond to the SPJR queries studied in the course *Bases de Données I*, where we took the named algebra perspective that uses $\bowtie$ and rename operations instead of Cartesian product.

1 Conjunctive Queries and Canonical Ground Rules

This section defines conjunctive queries and some other concepts that will be useful in later sections.

1.1 Conjunctive Queries

Syntax. A *conjunctive query* is a single Datalog rule such that all predicates in the rule body are EDB predicates. We will use the syntax of DLV[2] in this primer: variables start with uppercase letters, and constants start with lowercase letters. For example, `X` is a variable, and `x` is a constant. For reasons that will become apparent shortly, we will impose the following requirement:

> *Requirement of No-Name-Clashes:* A conjunctive query must not contain a variable that is letter-by-letter the same as some constant in the query. For example, if a query contains the constant `an`, then no variable in the query must be called `AN` or `An`.

An example of a conjunctive query is:

```
happy(X) :- knows(X,Person), owns(Person,iPad), owns(Person,iPod)        (q1)
```

Thus, `X` and `Person` are variables, while `iPad` and `iPod` are constants. In this primer, we will not terminate all rules with a period (.) as required in DLV.

Semantics. Let I be a database instance, and q a conjunctive query. The *answer* to q on I will be denoted[3] $eval(q, I)$ and is informally defined as follows:

> Repeatedly instantiate q, i.e., construct ground rules by replacing variables with constants. Whenever the body of such a ground rule is a subset of I, then add the rule head to the answer. In this case, we will also say that the rule head has been inferred from I by means of (a ground rule of) the query q.

For example, for the database $I :=$ {`knows(jeb,don), knows(don,jeb), knows(an,don), knows(ed,an), owns(don,iPad), owns(don,iPod), owns(jeb,iPod)`}, the answer to q_1 is:

$$eval(q_1, I) = \{\texttt{happy(jeb), happy(an)}\}.$$

We have that `happy(an)` belongs to the answer because q_1 can be instantiated by the following ground rule whose body is a subset of I:

```
happy(an) :- knows(an,don), owns(don,iPad), owns(don,iPod)
```

1.2 Freezing Variables

The *canonical ground rule* of a query q is obtained by changing the first letter of each variable in a lowercase letter. This operation of changing variables into constants is also called *freezing*. For example, the canonical ground rule of q_1 is as follows:

```
happy(x) :- knows(x,person), owns(person,iPad), owns(person,iPod)
```

Since we imposed the *Requirement of No-Name-Clashes*, no variable will be *"freezed"* into a constant that already occurs in the query. Therefore, the freezing operation can be naturally inverted. In this primer, the term *defrosting* will be used for this inverse operation.[4]

Remark 1.1. We now give a motivation for the *Requirement of No-Name-Clashes*. Without this requirement, it would be possible to write, for example, `answer(Z) :- r(Z,z)` and `answer(Z) :- r(z,Z)`. Since these two queries freeze into the same ground rule `answer(z) :- r(z,z)`, the "defrosting" operation would not be uniquely defined. □

[2]The DLV system is available at `http://www.dlvsystem.com/` and its syntax is described at `http://www.dlvsystem.com/html/DLV_User_Manual.html`.

[3]In *Bases de Données I*, this was denoted $[\![q]\!]^I$. I changed the notation in the current primer for readability reasons.

[4]The term *defrosting* is not used in the database literature. I introduced it in this primer to help the students' understanding.

By definition, canonical ground rules do not contain variables. Therefore, the body of a canonical ground rule is a set of EDB facts, which we will call the *canonical body*; the head is a single IDB fact which will be called the *canonical head*. Since a database instance was defined as a set of EDB facts (see the document entitled *A Datalog Primer*), a canonical body *is* actually a database instance. The picture is thus as follows:

$$\underbrace{\texttt{happy(x)}}_{\substack{\text{canonical}\\ \text{head of } q_1}} \texttt{ :- } \underbrace{\texttt{knows(x,person), owns(person,iPad), owns(person,iPod)}}_{\text{canonical body of } q_1}$$

Notations. In what follows, we will often write conjunctive queries as $\mathcal{H} :- \mathcal{B}$, where $\mathcal{H}$ and $\mathcal{B}$ are the head and body respectively. The canonical ground rule will often be denoted as $\mathcal{h} :- \mathcal{b}$, using lowercase letters. This notation is suggestive, because $\mathcal{h} :- \mathcal{b}$ is obtained from $\mathcal{H} :- \mathcal{B}$ by changing the first (uppercase) letter of each variable in lowercase. We will often use a symbol q to refer to such a query: the notation $q : \mathcal{H} :- \mathcal{B}$ is used to indicate that q is the conjunctive query with head $\mathcal{H}$ and body $\mathcal{B}$. Figure 1 summarizes our notations.

In the theoretical treatment, uppercase letters $X, Y, Z, X_1, Y_1, Z_1, \ldots$ are variables. Their corresponding constants are $x, y, z, x_1, y_1, z_1, \ldots$

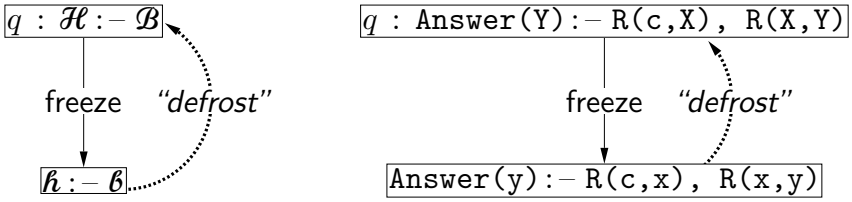

Figure 1: Representation *(left)* and illustration *(right)* of notations.

1.3 Querying Canonical Bodies

In the study of conjunctive queries, we will often execute a conjunctive query q on the canonical body of some conjunctive query q'. A particular case is where q is the same query as q'.

Here is a trivial observation:

> if q is a conjunctive query with canonical ground rule $\mathcal{h} :- \mathcal{b}$, then $eval(q, \mathcal{b})$ contains $\mathcal{h}$. Informally, when we execute a conjunctive query on its own canonical body, we can always infer its canonical head.

For example, let $\mathcal{b}_1 := \{$`knows(x,person), owns(person,iPad), owns(person,iPod)`$\}$, which is the canonical body of q_1. Then, `happy(x)` belongs to $eval(q_1, \mathcal{b}_1)$ because it can be inferred by means of the canonical ground rule of q_1, which is repeated next:

```
happy(x) :- knows(x,person), owns(person,iPad), owns(person,iPod)
```

It is easily verified that `happy(x)` is the only fact than can be inferred from $\mathcal{b}_1$ by means of q_1. However, in general, if we execute a conjunctive query on its own canonical body, the answer may contain more than one fact, as illustrated by the following example.

Example 1.1. Consider the following query:

```
happy(X) :- knows(X,Y), knows(Y,Z), knows(Z,X)        (q2)
```

whose canonical body is:

```
{ knows(x,y), knows(y,z), knows(z,x) }
```

Obviously, the canonical ground rule of q_2, which is given next, allows us to infer `happy(x)` from this canonical body:

```
happy(x) :- knows(x,y), knows(y,z), knows(z,x)
```

Now we claim that we can also infer `happy(y)`. Indeed, if we apply the replacement `X` $\mapsto$ `y`, `Y` $\mapsto$ `z`, `Z` $\mapsto$ `x` to q_2, we obtain the following ground rule:

```
happy(y) :- knows(y,z), knows(z,x), knows(x,y)
```

Note that the body of the latter ground rule is the canonical body of q_2, listed in a different order. Therefore, we can infer that the rule head `happy(y)` is true. It is an exercise for the student to verify that `happy(z)` is also true. Therefore, if we execute q_2 on its own canonical body, the answer is {`happy(x)`, `happy(y)`, `happy(z)`}. □

1.4 Conjunctive Queries in Other Languages

Conjunctive queries can of course be expressed in other languages. Assume a database schema $Knows[A, B]$ and $Owns[B, P]$. The query q_1 looks as follows in SQL:

```
SELECT A
FROM   Knows, Owns AS O1, Owns AS O2
WHERE  Knows.B = O1.B
AND    Knows.B = O2.B
AND    O1.P='iPad'
AND    O2.P='iPod'
```

In relational calculus:

$$\{x \mid \exists y\,(Knows(x, y) \wedge Owns(y, \texttt{iPad}) \wedge Owns(y, \texttt{iPod}))\}.$$

In relational algebra:

$$\pi_A(Knows \bowtie \pi_B(\sigma_{P=\texttt{iPad}}(Owns)) \bowtie \pi_B(\sigma_{P=\texttt{iPad}}(Owns))).$$

Exercise 1.1. Argue that every conjunctive query in Datalog syntax can be expressed as a relational calculus query that uses only $\exists$ and $\wedge$ (and parentheses). Argue that every conjunctive query in Datalog can be expressed in relational algebra without using union or negation. □

In database theory, the study of conjunctive queries often uses Datalog syntax because this simplifies the technical treatment. Indeed, some results (Proposition 2, Theorem 1) can be more elegantly stated and proved in Datalog syntax, compared to, for example, relational algebra.

2 Query Containment

In this section, we introduce the containment problem for conjunctive queries, and argue that it is of practical importance in database systems. We then show some elegant solutions to this problem.

2.1 The Containment Problem

Consider the following query:

```
happy(U) :- knows(U,W), owns(W,iPad)
```
(q_3)

It can be easily verified that for every database instance I, the following holds true: if we can infer `happy(c)`, for some person `c`, by means of q_1, then we can also infer `happy(c)` by means of q_3. Informally, if a person `c` knows someone owning both an iPad and an iPod (cf. q_1), then `c` necessarily knows someone owning an iPad (cf. q_3). This relationship between q_1 and q_3 will be denoted $q_1 \sqsubseteq q_3$.

Formally, if q_1 and q_2 are two conjunctive queries, then we write $q_1 \sqsubseteq q_2$ if for every database instance I, we have $eval(q_1, I) \subseteq eval(q_2, I)$. Note that the symbol $\subseteq$ is standard set inclusion. If $q_1 \sqsubseteq q_2$ holds true, then we also say that q_1 *is contained in* q_2.

The CONTAINMENT PROBLEM for conjunctive queries is the following problem:

INPUT: Two conjunctive queries q_1 and q_2.

QUESTION: Is q_1 contained in q_2?

This problem is of fundamental interest in database systems, among others, because it allows us to verify that a complex query can be simplified. Say that two queries q_1 and q_2 are *equivalent*, denoted $q_1 \equiv q_2$, if for every database instance I, we have $eval(q_1, I) = eval(q_2, I)$. For example, in relational algebra, it is easily verified that $E := \pi_{A=B}(R \cup \pi_{sort(R)}(R \bowtie S))$ is equivalent to simply $\pi_{A=B}(R)$, which is a significant simplification in practice (because joins are expensive). Now it is obvious to see that $q_1 \equiv q_2$ holds true if and only if both $q_1 \sqsubseteq q_2$ and $q_2 \sqsubseteq q_1$ hold true. Therefore, to test whether two queries are equivalent, it is sufficient to solve two instances of the CONTAINMENT PROBLEM.

Importantly, we cannot practically solve the CONTAINMENT PROBLEM by a procedure that tests $eval(q_1, I) \subseteq eval(q_2, I)$ for *every* database instance I. Indeed, in case that $eval(q_1, I) \subseteq eval(q_2, I)$ holds true, such a procedure would never terminate, as there are infinitely many database instances.

We now show a really awesome result: to test $q_1 \sqsubseteq q_2$ for conjunctive queries, it suffices to consider *only one* database instance, which is the canonical body of q_1. If the answer to q_1 is included in the answer to q_2 on this canonical body, then the answer to q_1 is included in the answer to q_2 on any database instance. This is expressed by the following proposition.

Proposition 1. *Let q_1 and q_2 be conjunctive queries. Let the canonical ground rule of q_1 be $\boldsymbol{h}_1 :- \boldsymbol{b}_1$. Then,*

1. *if $eval(q_1, \boldsymbol{b}_1) \subseteq eval(q_2, \boldsymbol{b}_1)$, then $q_1 \sqsubseteq q_2$; and*
2. *if $eval(q_1, \boldsymbol{b}_1) \not\subseteq eval(q_2, \boldsymbol{b}_1)$, then $q_1 \not\sqsubseteq q_2$.*

In the next section, we will actually show a slightly stronger and more practical result:

Proposition 2. *Let q_1 and q_2 be conjunctive queries. Let the canonical ground rule of q_1 be $\boldsymbol{h}_1 :- \boldsymbol{b}_1$. Then,*

1. *if $\boldsymbol{h}_1 \in eval(q_2, \boldsymbol{b}_1)$, then $q_1 \sqsubseteq q_2$; and*
2. *if $\boldsymbol{h}_1 \notin eval(q_2, \boldsymbol{b}_1)$, then $q_1 \not\sqsubseteq q_2$.*

The difficult part to prove is the first item of Proposition 2.

Exercise 2.1 (Easy)**.** Prove the second item of Proposition 1, and the second item of Proposition 2. □

Exercise 2.2 (Easy)**.** In Section 2.3, we will prove the first item of Proposition 2. Argue that the first item of Proposition 1 is an immediate corollary of the first item of Proposition 2. □

2.2 Detailed Illustration of Proposition 2

We illustrate Proposition 2 by an example. To this end, we recall our query q_1:

```
happy(X) :- knows(X,Person), owns(Person,iPad), owns(Person,iPod)        (q1)
```

The canonical ground rule of q_1 is composed as follows:

$$\underbrace{\texttt{happy(x)}}_{\substack{\text{canonical head of } q_1, \\ \text{called } \boldsymbol{h}_1}} \texttt{ :- } \underbrace{\texttt{knows(x,person), owns(person,iPad), owns(person,iPod)}}_{\text{canonical body of } q_1\text{, called } \boldsymbol{b}_1 \text{ from here on}}$$

We next recall our query q_3:

```
happy(U) :- knows(U,W), owns(W,iPad)        (q3)
```

Let $\boldsymbol{b}_1$ be the canonical body of q_1, and let $\boldsymbol{h}_1$ be the canonical head of q_1 (as indicated above). The replacement $\texttt{U} \mapsto \texttt{x}$, $\texttt{W} \mapsto \texttt{person}$ instantiates q_3 as follows:

```
happy(x) :- knows(x,person), owns(person,iPad)
```

Since the body of this instantiation is a subset of $\boldsymbol{b}_1$, we can infer its head, that is:

$$\texttt{happy(x)} \in eval(q_3, \boldsymbol{b}_1).$$

Since `happy(x)` $= \boldsymbol{\hbar}_1$, we have $\boldsymbol{\hbar}_1 \in \mathit{eval}(q_3, \boldsymbol{\mathcal{b}}_1)$. It follows from Proposition 2 that $q_1 \sqsubseteq q_3$.

We now argue why $q_1 \sqsubseteq q_3$ must be true *without using Proposition 2*. To this end, let I be an arbitrary database instance, and assume that for some constant `a`, we have that `happy(a)` is an answer to q_1 on I, that is, `happy(a)` $\in \mathit{eval}(q_1, I)$. Then, for some constant `b`, there must be an instantiation of q_1 that looks as follows:

$$\underbrace{\texttt{happy(a)}}_{\substack{\text{inferred}\\ \text{head}}} \texttt{ :- } \underbrace{\texttt{knows(a,b), owns(b,iPad), owns(b,iPod)}}_{\text{subset of } I\text{, called } \boldsymbol{\mathcal{b}}_1' \text{ from here on}}$$

Now the replacement `U` $\mapsto$ `a`, `W` $\mapsto$ `b` instantiates q_3 as follows:

```
happy(a) :- knows(a,b), owns(b,iPad)
```

Since the body of this instantiation is a subset of I (because it is a subset of $\boldsymbol{\mathcal{b}}_1'$, which itself is a subset of I), we can infer its head, that is:

$$\texttt{happy(a)} \in \mathit{eval}(q_3, I).$$

Since I and `a` were chosen arbitrarily, it is correct to conclude $q_1 \sqsubseteq q_3$. Note that if our argumentation of $q_1 \sqsubseteq q_3$ would have used `x` instead of `a`, and `person` instead of `b`, then it would coincide with our illustration of Proposition 2 in the beginning of this section.

Exercise 2.3. Here are two conjunctive queries.

q_4 : `answer(X,U) :- r(X,R,S), r(V,S,U), r(X,Y,Z), r(Y,Z,U), r(U,Y,W)`

q_5 : `answer(X,V) :- r(X,Y,Z), r(X,Y,V), r(Y,Z,V), r(V,Y,W)`

Use Proposition 2 and the DLV system to verify whether either query is contained in the other.

Solution. Freezing q_5 gives us:

$$\underbrace{\texttt{answer(x,v)}}_{\substack{\text{canonical}\\ \text{head of } q_5,\\ \text{called } \boldsymbol{\hbar}_5}} \texttt{ :- } \underbrace{\texttt{r(x,y,z), r(x,y,v), r(y,z,v), r(v,y,w)}}_{\text{canonical body of } q_5\text{, called } \boldsymbol{\mathcal{b}}_5 \text{ from here on}}$$

The result $\mathit{eval}(q_4, \boldsymbol{\mathcal{b}}_5)$ is computed by means of the following DLV program.

```
answer(X,U) :- r(X,R,S), r(V,S,U), r(X,Y,Z), r(Y,Z,U), r(U,Y,W).
r(x,y,z). r(x,y,v). r(y,z,v). r(v,y,w).
```

Running this program in DLV yields the following output:

```
DLV [build BEN/Dec 17 2012   gcc 4.6.1]
{r(x,y,z), r(x,y,v), r(y,z,v), r(v,y,w), answer(x,v)}
```

Since `answer(x,v)` is in the output, Proposition 2 tells us that $q_5 \sqsubseteq q_4$. We leave it is an exercise to test whether or not $q_4 \sqsubseteq q_5$ holds true. □

2.3 Proof of Proposition 2

The proof of Proposition 2 essentially generalizes the example of Section 2.2 to an arbitrary conjunctive query. Since the proof provides interesting insights into conjunctive queries, we spell it out in the current section.

We will need the notion of *valuation*, which is a formalization of what we called "instantiation" or "replacement" so far. A valuation μ for a conjunctive query q is a function that maps every variable of q to a constant, for example, $\mu = \{X \mapsto a, Y \mapsto b, Z \mapsto c\}$.

Informally, a valuation assigns a *value* (i.e., a constant) to each variable. A valuation for a conjunctive query q can be naturally extended to the head and the body of that query. For example, the above valuation μ maps $r(X, X, Y, c, d)$ to $r(a, a, b, c, d)$. Note that this valuation did not change the rightmost constants c and d. In general, it is understood that a valuation maps every constant to itself. For another example, if $B = \{P(X, Y), P(Y, Z)\}$, then $\mu(B) = \{P(a, b), P(b, c)\}$. Using the notion of valuation, the semantics of conjunctive queries can be stated as follows (in a box, because it is important).

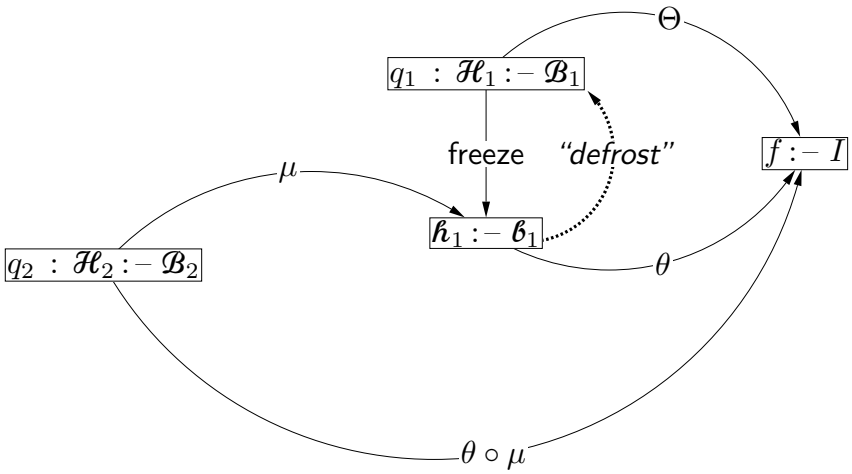

Figure 2: Visualization of the proof [of the first item] of Proposition 2.

Semantics. Then, for every query $q : \mathcal{H} :- \mathcal{B}$, database instance I, and fact f, the following two statements are equivalent (by definition):

- $f \in \mathit{eval}(q, I)$; and
- there exists a valuation μ for q such that $\mu(\mathcal{B}) \subseteq I$ and $\mu(\mathcal{H}) = f$.

We will now give a formal proof of the first item in Proposition 2. The flow of the proof is illustrated in Fig. 2. Let q_1 and q_2 be as follows:

$$\begin{aligned} q_1 &: \mathcal{H}_1 :- \mathcal{B}_1 \\ q_2 &: \mathcal{H}_2 :- \mathcal{B}_2 \end{aligned}$$

Assume that the canonical ground rule of q_1 is as follows:

$$\boldsymbol{h}_1 :- \boldsymbol{b}_1$$

Recall that the canonical ground rule of q_1 is obtained from q_1 by changing the first letter of each variable in a lowercase letter. Our hypothesis in Proposition 2 is:

$$\boldsymbol{h}_1 \in \mathit{eval}(q_2, \boldsymbol{b}_1). \tag{1}$$

We need to show $q_1 \sqsubseteq q_2$.

To show $q_1 \sqsubseteq q_2$, we imagine an arbitrary database I and an arbitrary fact f such that

$$f \in \mathit{eval}(q_1, I). \tag{2}$$

It suffices to show $f \in \mathit{eval}(q_2, I)$. The reason why this suffices is that I and f are chosen arbitrarily.[5] Let us first spell out the meaning of (2): there exists a valuation Θ for q_1 such that

$$\Theta(\mathcal{H}_1) = f \text{ and } \Theta(\mathcal{B}_1) \subseteq I. \tag{3}$$

In other words, (2) means that we can instantiate q_1 into a ground rule $f :- \Theta(\mathcal{B}_1)$ such that $\Theta(\mathcal{B}_1) \subseteq I$. Θ maps uppercase variables $X, Y, Z, \ldots$ to constants that occur in the database. Now we use a small notational trick: we define θ as the mapping that is the same as Θ except that it applies to the lowercase strings $x, y, z, \ldots$ That is, for every variable X in q_1, we define $\theta(x) := \Theta(X)$. Furthermore, for every constant c in q_1, we define $\theta(c) := c$. We note here that θ is a well-defined function because of the *Requirement of No-Name-Clashes*.

[5] This reasoning should be familiar to the student. Remember, for example, that $A \subseteq B$ is often proved as follows: imagine an arbitrary element x in A, and prove that x is also in B.

Remark 2.1. Without the *Requirement of No-Name-Clashes*, it would be possible to write, for example, q : `happy(MisterX)` :– `knows(MisterX, misterX)`. Now assume we have Θ(`MisterX`) = `jeb`. Our definition would require both θ(`misterx`) = `jeb` and θ(`misterx`) = `misterx`, and therefore θ would not be a well-defined function. See also Remark 1.1. □

It is also correct (and insightful) to think of θ as the composition "Θ after defrost," as suggested in Fig. 2.

Since $\boldsymbol{\mathcal{h}}_1 :- \boldsymbol{\mathcal{b}}_1$ is obtained from $\mathcal{H}_1 :- \mathcal{B}_1$ by replacing $X, Y, Z, \ldots$ with $x, y, z, \ldots$, it is easily verified that $\theta(\boldsymbol{\mathcal{h}}_1) = \Theta(\mathcal{H}_1)$ and $\theta(\boldsymbol{\mathcal{b}}_1) = \Theta(\mathcal{B}_1)$. Then, by (3), we obtain the following:

$$\theta(\boldsymbol{\mathcal{h}}_1) = f \text{ and } \theta(\boldsymbol{\mathcal{b}}_1) \subseteq I. \tag{4}$$

We now spell out the meaning of our hypothesis, which was given by (1): there exists a valuation μ for q_2 such that

$$\mu(\mathcal{H}_2) = \boldsymbol{\mathcal{h}}_1 \text{ and } \mu(\mathcal{B}_2) \subseteq \boldsymbol{\mathcal{b}}_1. \tag{5}$$

If we apply θ on (5), we obtain:

$$\theta\left(\mu(\mathcal{H}_2)\right) = \theta(\boldsymbol{\mathcal{h}}_1) \text{ and } \theta\left(\mu(\mathcal{B}_2)\right) \subseteq \theta(\boldsymbol{\mathcal{b}}_1). \tag{6}$$

Now let $\theta \circ \mu$ be the function such that for every X, we have $(\theta \circ \mu)(X) := \theta(\mu(X))$. Note that $\circ$ denotes the function composition operation, which should be familiar to the student. It is easily verified that $\theta \circ \mu$ is a valuation. From (4) and (6), we obtain:

$$(\theta \circ \mu)(\mathcal{H}_2) = f \text{ and } (\theta \circ \mu)(\mathcal{B}_2) \subseteq I.$$

In other words, it is possible to instantiate q_2 into a ground rule $f :- (\theta \circ \mu)(\mathcal{B}_2)$ such that $(\theta \circ \mu)(\mathcal{B}_2) \subseteq I$. This tells us that $f \in \mathit{eval}(q_2, I)$, which concludes the proof.

2.4 The Homomorphism Theorem

In this section, we will state a result that in database theory is known as the Homomorphism Theorem. It is basically a different way for stating Proposition 2.

Look again at Fig. 2, which illustrates how the hypothesis $\boldsymbol{\mathcal{h}}_1 \in \mathit{eval}(q_2, \boldsymbol{\mathcal{b}}_1)$ (which is Equation (1)) leads to the conclusion $q_1 \sqsubseteq q_2$. Consider the mapping "defrost after μ," which we will call Γ. Informally, the Homomorphism Theorem can be discovered in the following three steps:

1. Let X be a variable that occurs in $\mathcal{B}_2$. If μ maps X to some constant u, then Γ maps X to the variable U. Recall here that U *freezes* into u, and therefore u *defrosts* into U. Since Γ maps X to a variable, it is not a valuation (because valuations map to constants, not to variables). *Substitutions*, on the other hand, can map a variable to a variable.

2. By "defrosting" (5), we obtain that for the substitution Γ, it holds that $\Gamma(\mathcal{B}_2) \subseteq \mathcal{B}_1$ and $\Gamma(\mathcal{H}_2) = \mathcal{H}_1$. Such a substitution will be called a *homomorphism* from q_2 to q_1. That is, a homomorphism from q_2 to q_1 is a substitution that maps the body of q_2 into the body of q_1, and maps the head of q_2 to the head of q_1.

3. Now the Homomorphism Theorem states what you may already expect from Fig. 2:

   $q_1 \sqsubseteq q_2$ if and only if there is a homomorphism from q_2 to q_1.

   The nice thing about this theorem is that it talks only about (the bodies and heads of) q_1 and q_2. In particular, unlike Proposition 2, the Homomorphism Theorem does not refer to the frozen body $\boldsymbol{\mathcal{b}}_1$ or the frozen head $\boldsymbol{\mathcal{h}}_1$ of q_1.

Example 2.1. Consider once more the following two queries:

q_1 : `happy(X)` :– `knows(X,Person), owns(Person,iPad), owns(Person,iPod)`

q_3 : `happy(U)` :– `knows(U,W), owns(W,iPad)`

The substitution $\Gamma := \{$`U` $\mapsto$ `X`, `W` $\mapsto$ `Person`$\}$ is a homomorphism from q_3 to q_1. Indeed, if we apply that substitution to q_3, we obtain the head of q_1 together with a subset of the body of q_1:

$\Gamma(q_3)$: `happy(X)` :– `knows(X,Person), owns(Person,iPad)`

Then, the Homomorphism Theorem tells us that $q_1 \sqsubseteq q_3$.

There exists no homomorphism from q_1 to q_3, because no substitution can map the atom `owns(Y,iPod)` to an atom of q_1. Then, the Homomorphism Theorem tells us $q_3 \not\sqsubseteq q_1$. □

We will now introduce the Homomorphism Theorem in a formal way.

A *substitution* for a conjunctive query q is a function that maps every variable of q to either a variable or a constant. It is also understood that a substitution maps every constant to itself. A valuation is a special case of a substitution, i.e., every valuation is a substitution.

Theorem 1 (Homomorphism Theorem)**.** *Let* $q_1 : \mathcal{H}_1 :- \mathcal{B}_1$ *and* $q_2 : \mathcal{H}_2 :- \mathcal{B}_2$ *be two conjunctive queries. Then,*

$$q_1 \sqsubseteq q_2 \qquad \textit{if and only if} \qquad \begin{array}{l}\textit{there exists a substitution } \Gamma \textit{ for } q_2 \textit{ such}\\ \textit{that } \Gamma(\mathcal{B}_2) \subseteq \mathcal{B}_1 \textit{ and } \Gamma(\mathcal{H}_2) = \mathcal{H}_1.\end{array}$$

Such a substitution Γ*, if it exists, is also called a* homomorphism from q_2 to q_1.

The implication $\Longrightarrow$ immediately follows from the technical treatment in Section 2.3, and can be read off from Fig. 2. Indeed, assume $q_1 \sqsubseteq q_2$, which is the situation depicted in that figure. The homomorphism Γ is given by "defrost after μ." The proof of the $\Longleftarrow$-direction is left as an exercise.

Exercise 2.4. Consider the queries q_4 and q_5 of Exercise 3.1, where it was shown that $q_5 \sqsubseteq q_4$. By Theorem 1 there exists a homomorphism from q_4 to q_5. Give this homomorphism. □

Exercise 2.5. An *endomorphism* is a homomorphism from a query q to itself. Verify that for the query q_0 : `answer(X) :- r(X,Y), r(Y,X), r(X,Z)`, the substitution $\Gamma_0 = \{\mathtt{X} \mapsto \mathtt{X},\ \mathtt{Y} \mapsto \mathtt{Y},\ \mathtt{Z} \mapsto \mathtt{Y}\}$ is an endomorphism. Can you explain why Γ_0 is important in the perspective of query optimization? □

Exercise 2.6. In Example 1.1, we saw that the execution of a conjunctive query on its own canonical body can yield a query answer with cardinality > 1. If this happens, what can we say about the existence of homomorphisms?

Solution. Consider a conjunctive query $q : answer(X) :- \mathcal{B}$ such that $|eval(q, \boldsymbol{\mathcal{b}})| > 1$, where $\boldsymbol{\mathcal{b}}$ be the canonical body of q. For simplicity, assume that q contains no constants. Since $answer(x) \in eval(q, \boldsymbol{\mathcal{b}})$ is obvious, there is a constant y such that $answer(y) \in eval(q, \boldsymbol{\mathcal{b}})$ and $y \neq x$. Therefore, there is a valuation μ for q such that $\mu(\mathcal{B}) \subseteq \boldsymbol{\mathcal{b}}$ and $\mu(X) = y$. Using our hypothesis that q contains no constants, we can conclude that there is a substitution Γ for q such that $\Gamma(\mathcal{B}) \subseteq \mathcal{B}$ and $\Gamma(X) = Y$; note that Γ = "defrost after μ." Now consider the query:

$$q' : answer(Y) :- \Gamma(\mathcal{B}).$$

Clearly, Γ is a homomorphism from q to q' (and therefore $q' \sqsubseteq q$ by Theorem 1). An example is given next.

```
%%% Let q be as follows.
answer(X) :- r(X,Y), r(Y,X), r(Y,Y).
%%% The substitution { X↦Y, Y↦Y } gives the following query called q'.
answer(Y) :- r(Y,Y).
```

Verify that in the preceding example, there exists no homomorphism from q' to q. The substitution in this example maps the body $\{\mathtt{r(X,Y)},\ \mathtt{r(Y,X)},\ \mathtt{r(Y,Y)}\}$ to a strict subset of that body.[6]

Example 1.1 illustrates the case $\Gamma(\mathcal{B}) = \mathcal{B}$, i.e., $\Gamma(\mathcal{B})$ is not a strict subset of $\mathcal{B}$. Then Γ must necessarily map distinct variables to distinct variables, and therefore Γ is a bijection. In that case, q and q' are the same up to a renaming of variables. □

Exercise 2.7. Give a proof of Theorem 1.

Solution.

Proof. $\boxed{\Longrightarrow}$ Assume $q_1 \sqsubseteq q_2$. By Proposition 2, we have $\boldsymbol{\mathcal{h}}_1 \in eval(q_2, \boldsymbol{\mathcal{b}}_1)$, where $\boldsymbol{\mathcal{h}}_1 :- \boldsymbol{\mathcal{b}}_1$ is the canonical ground rule of q_1. That is, there exists a valuation μ for q_2 such that $\mu(\mathcal{H}_2) = \boldsymbol{\mathcal{h}}_1$ and $\mu(\mathcal{B}_2) \subseteq \boldsymbol{\mathcal{b}}_1$. Let Γ be the substitution for q_2 such that for every variable X in q_2, for every variable Y in

[6] A is a *strict subset* of B if both $A \subseteq B$ and $A \neq B$.

q_1, we have $\Gamma(X) = Y$ if $\mu(X) = y$. Informally, Γ maps X to the variable uppercase-Y if μ maps X to the corresponding constant lowercase-y. It is now easily verified that $\Gamma(\mathcal{B}_2) \subseteq \mathcal{B}_1$ and $\Gamma(\mathcal{H}_2) = \mathcal{H}_1$.

$\boxed{\Longleftarrow}$ Assume there is a substitution Γ for q_2 such that $\Gamma(\mathcal{B}_2) \subseteq \mathcal{B}_1$ and $\Gamma(\mathcal{H}_2) = \mathcal{H}_1$. Let μ be the valuation such that for every variable X in q_2, for every variable Y in q_1, we have $\mu(X) = y$ if $\Gamma(X) = Y$. Informally, μ is "freeze after Γ." Clearly, $\mu(\mathcal{B}_2) \subseteq \mathcal{b}_1$ and $\mu(\mathcal{H}_2) = \mathcal{h}_1$. Therefore, $\mathcal{h}_1 \in eval(q_2, \mathcal{b}_1)$. By Proposition 2, $q_1 \sqsubseteq q_2$. □

3 Minimization of Conjunctive Queries

Query minimization is an important step in query optimization. A conjunctive query with n atoms in its body, say $R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$, can be easily implemented by joining n relations (which requires $n-1$ binary joins), followed by zero or more selections and one projection. Since joins are costly operations, it is significant to minimize the number n of atoms in the body. In this section, we will study this minimization problem.

3.1 Minimization

Students are encouraged to make Exercise 2.5 before reading on, because that exercise illustrates what will happen in this section. We start with another example.

Example 3.1. Let

q_1 : `answer(X) :- r(X,Y), r(Y,X), r(X,U), r(U,V), r(V,W), r(W,X)`

q_2 : `answer(U) :- r(U,W), r(W,U)`

If we execute the following DLV program

```
r(u,w). r(w,u).
answer(X) :- r(X,Y), r(Y,X), r(X,U), r(U,V), r(V,W), r(W,X).
```

we get the answer {`answer(u), answer(w)`}. Since the canonical head of q_2 belongs to this answer, Proposition 2 allows us to conclude that $q_2 \sqsubseteq q_1$. Then, by Theorem 1, there must be a homomorphism from q_1 to q_2. Before reading further, the student is encouraged to find this homomorphism.

It is easily verified that the substitution $\mu := \{\mathtt{X} \mapsto \mathtt{U}, \mathtt{Y} \mapsto \mathtt{W}, \mathtt{U} \mapsto \mathtt{W}, \mathtt{V} \mapsto \mathtt{U}, \mathtt{W} \mapsto \mathtt{W}\}$ is a homomorphism from q_1 to q_2. Furthermore, the substitution $\theta := \{\mathtt{U} \mapsto \mathtt{X}, \mathtt{W} \mapsto \mathtt{Y}\}$ is a homomorphism from q_2 to q_1, and therefore, by Theorem 1, $q_1 \sqsubseteq q_2$.

Now consider the composition $\theta \circ \mu$, which is as follows:

$$\begin{aligned} \theta \circ \mu &= \{\mathtt{X} \mapsto \theta(\mathtt{U}), \mathtt{Y} \mapsto \theta(\mathtt{W}), \mathtt{U} \mapsto \theta(\mathtt{W}), \mathtt{V} \mapsto \theta(\mathtt{U}), \mathtt{W} \mapsto \theta(\mathtt{W})\} \\ &= \{\mathtt{X} \mapsto \mathtt{X}, \mathtt{Y} \mapsto \mathtt{Y}, \mathtt{U} \mapsto \mathtt{Y}, \mathtt{V} \mapsto \mathtt{X}, \mathtt{W} \mapsto \mathtt{Y}\}. \end{aligned}$$

If we apply $\theta \circ \mu$ to q_1, we obtain the following query (duplicates have been removed from the rule body):

q_1' : `answer(X) :- r(X,Y), r(Y,X)`

From our construction, it follows that $\theta \circ \mu$ is a homomorphism from q_1 to q_1'. Moreover, the identity substitution $\{\mathtt{X} \mapsto \mathtt{X}, \mathtt{Y} \mapsto \mathtt{Y}\}$ is a homomorphism from q_1' to q_1. By Theorem 1, we have $q_1' \sqsubseteq q_1$ and $q_1 \sqsubseteq q_1'$, and therefore $q_1' \equiv q_1$. In conclusion, we have simplified q_1 into q_1'. □

In general, given a conjunctive query q, an important task is to find a query that is equivalent to q and that has a body of minimal cardinality. QUERY MINIMIZATION is the following task:

INPUT: A conjunctive query $q : \mathcal{H} :- \mathcal{B}$.

QUESTION: Find an equivalent conjunctive query with a cardinality-minimal body.

Example 3.1 suggests that, given a conjunctive query q, an equivalent query with a cardinality-minimal body can be obtained by deleting atoms from the body of q. We will show next that this suggestion is indeed correct.

The cardinality of a set S is denoted by $|S|$. Consider a conjunctive query $q : \mathcal{H} :- \mathcal{B}$. Now let $q_{\min} : \mathcal{H}_{\min} :- \mathcal{B}_{\min}$ be a conjunctive query equivalent to q with a cardinality-minimal body. That is, $q_{\min} \equiv q$ and every conjunctive query q' equivalent to q has a body with at least $|\mathcal{B}_{\min}|$ atoms.

It can be easily seen that such a query $q_{\min}$ must exist, even though we may not know yet how to find it. We show next that such a query $q_{\min}$ can always be obtained by deleting zero, one, or more atoms from the body of q.

Since $q_{\min} \equiv q$, we have $q \sqsubseteq q_{\min}$ and $q_{\min} \sqsubseteq q$.

By Theorem 1, there exists some homomorphism μ from q to $q_{\min}$, and another homomorphism θ from $q_{\min}$ to q. This situation is depicted in Fig. 3.

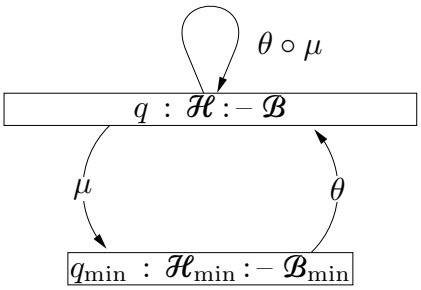

Figure 3: The composition $\theta \circ \mu$ is an endomorphism that minimizes q.

Thus,

$$\mu(\mathcal{B}) \subseteq \mathcal{B}_{\min} \quad (7)$$
$$\mu(\mathcal{H}) = \mathcal{H}_{\min} \quad (8)$$
$$\theta(\mathcal{B}_{\min}) \subseteq \mathcal{B} \quad (9)$$
$$\theta(\mathcal{H}_{\min}) = \mathcal{H} \quad (10)$$

For the composition $\theta \circ \mu$, we have the following:

- For the bodies, we have $(\theta \circ \mu)(\mathcal{B}) = \theta(\mu(\mathcal{B}))$. It follows from (7) that $(\theta \circ \mu)(\mathcal{B}) \subseteq \theta(\mathcal{B}_{\min})$. Consequently, by (9), we have $(\theta \circ \mu)(\mathcal{B}) \subseteq \mathcal{B}$.
- For the heads, we have $(\theta \circ \mu)(\mathcal{H}) = \theta(\mu(\mathcal{H}))$. It follows from (8) that $(\theta \circ \mu)(\mathcal{H}) = \theta(\mathcal{H}_{\min})$. Consequently, by (10), we have $(\theta \circ \mu)(\mathcal{H}) = \mathcal{H}$.

Now consider the following conjunctive query q':

$$q' : \mathcal{H} :- \mathcal{B}' \text{ with } \mathcal{B}' := (\theta \circ \mu)(\mathcal{B})$$

It is easily verified that $\theta \circ \mu$ is a homomorphism from q to q'. Indeed, $(\theta \circ \mu)(\mathcal{B}) \subseteq \mathcal{B}'$ holds true by construction, and $(\theta \circ \mu)(\mathcal{H}) = \mathcal{H}$ holds true by the second item above. Therefore, by Theorem 1, we have $q' \sqsubseteq q$. Moreover, the substitution that maps every variable to itself is a homomorphism from q' to q, because *(i)* the body of q' is a subset of the body of q (see first item above), and *(ii)* q' and q have the same head. Therefore, by Theorem 1, we have $q \sqsubseteq q'$.

It is now correct to conclude $q \equiv q'$, where q' was obtained from q by deleting zero, one, or more atoms from the body. So far so good. But the most important thing remains to be shown, namely that $|\mathcal{B}'| = |\mathcal{B}_{\min}|$. Note that $|\mathcal{B}'| < |\mathcal{B}_{\min}|$ is impossible, because of our hypothesis that $q_{\min}$ has a cardinality-minimal body.

Well, from $(\theta \circ \mu)(\mathcal{B}) \subseteq \theta(\mathcal{B}_{\min})$ and $\mathcal{B}' = (\theta \circ \mu)(\mathcal{B})$, it follows $\mathcal{B}' \subseteq \theta(\mathcal{B}_{\min})$. It can be easily verified that $|\theta(\mathcal{B}_{\min})| \leq |\mathcal{B}_{\min}|$, because the application of a substitution cannot increase the number of atoms. Since a subset cannot have more elements than its superset, we have $|\mathcal{B}'| \leq |\theta(\mathcal{B}_{\min})| \leq |\mathcal{B}_{\min}|$. Therefore, the query q' is what we were looking for: a query equivalent to q with a cardinality-minimal body that can be obtained by deleting atoms from the body of q.

3.2 Uniqueness of Minimal Conjunctive Queries

We now show an easy but interesting result: every conjunctive query with a cardinality-minimal body is unique up to a renaming of its variables. This is sometimes also stated as follows: if two minimized conjunctive queries are equivalent, then they are *isomorphic*.

Assume two equivalent conjunctive queries $q_1 \equiv q_2$, both with cardinality-minimal bodies. Let $q_1 : \mathcal{H}_1 :- \mathcal{B}_1$ and $q_2 : \mathcal{H}_2 :- \mathcal{B}_2$. From the reasoning in Section 3.1, it follows $|\mathcal{B}_1| = |\mathcal{B}_2|$. By Theorem 1, there exists a homomorphism μ from q_1 to q_2, and a homomorphism θ from q_2 to q_1. As argued in Section 3.1, $\theta \circ \mu$ is a homomorphism from q_1 to itself, that is, $(\theta \circ \mu)(\mathcal{H}_1) = \mathcal{H}_1$ and $(\theta \circ \mu)(\mathcal{B}_1) \subseteq \mathcal{B}_1$. Since q_1 is minimized, it must be the case that $(\theta \circ \mu)(\mathcal{B}_1) = \mathcal{B}_1$. Therefore, every variable that occurs in $\mathcal{B}_1$ must also occur in $(\theta \circ \mu)(\mathcal{B}_1)$. But this implies that both μ and θ must be injective, i.e., they cannot map distinct variables to a same variable. Therefore, q_1 and q_2 are the same up to a renaming of variables.

Example 3.2. Consider again the queries of Example 3.1, after minimization.

$$
\begin{array}{rcl}
q_1' & : & \texttt{answer(X)} :- \texttt{r(X,Y), r(Y,X)} \\
q_2 & : & \texttt{answer(U)} :- \texttt{r(U,W), r(W,U)}
\end{array}
$$

These two queries are indeed the same up to a renaming of their variables. □

4 Complexity

In this section, we show that the study of conjunctive queries is relevant also outside the scope of database systems. Indeed, it turns out that many notorious problems in computer science can be formulated in terms of containment of conjunctive queries.

4.1 Beyond Database Systems

We will establish a relationship between containment of conjunctive queries and 3-COLORABILITY, which is the following problem:

INPUT: An undirected graph.

QUESTION: Is the graph 3-colorable, i.e., is it possible to color the vertices with three colors (red, blue, green) such that no two adjacent vertices have the same color?

An instance of this problem is the undirected graph of Fig. 4. We claim that 3-COLORABILITY is the same as the CONTAINMENT PROBLEM for conjunctive queries. Consider the following two queries, called q_{graph} and q_{rgb}, using DLV syntax.

% The query q_{graph} is:

```
coloring :- e(A,B), e(B,C), e(C,D), e(D,E), e(E,F), e(F,G), e(G,H), e(H,A), e(D,H),
            e(B,F), e(I,C), e(C,J), e(J,G), e(G,I), e(A,I), e(B,H), e(E,J), e(A,E).
```

body of q_{graph}, called $\mathcal{B}_{\text{graph}}$ from here on

% The query q_{rgb} is:

```
coloring :- e(red,green), e(green,blue), e(blue,red),
            e(green,red), e(blue,green), e(red,blue).
```

body of q_{rgb}, called $\mathcal{B}_{\text{rgb}}$ from here on

The predicate `coloring` has arity 0; it acts like a propositional variable: the answer to q_{graph} or q_{rgb} can be {`coloring`} or {}, which can be interpreted as true and false respectively. The query q_{graph} encodes the undirected graph of Fig. 4: every undirected edge {`X`, `Y`} is encoded as either `e(X,Y)` or `e(Y,X)` (either choice is fine). The query q_{rgb} encodes all ordered pairs of two distinct colors among red, green, blue. Now we ask the question:

Is q_{rgb} contained in q_{graph}, i.e., do we have $q_{\text{rgb}} \sqsubseteq q_{\text{graph}}$?

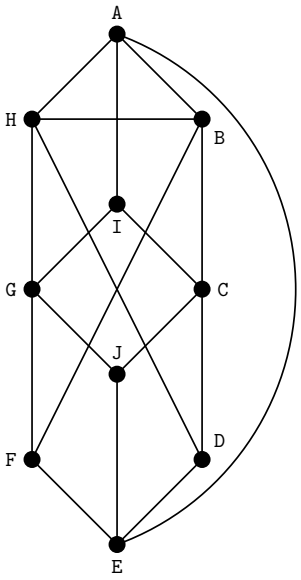

Figure 4: Undirected graph.

By Theorem 1, this question is the same as:

> Is there a substitution for q_{graph} that maps $\mathcal{B}_{\text{graph}}$ into $\mathcal{B}_{\text{rgb}}$? In other words, is there a homomorphism from q_{graph} to q_{rgb}?

Note that we can ignore rule heads, since they are the same in both queries. We claim that if such a substitution exists, then the graph of Fig. 4 is 3-colorable. To this end, assume that Θ is a substitution that maps $\mathcal{B}_{\text{graph}}$ into $\mathcal{B}_{\text{rgb}}$. Note that Θ is actually a valuation, because it maps every variable in `A,B,C,...,J` to a constant among `red, green, blue`. This valuation cannot map two adjacent vertices to the same color, because $\mathcal{B}_{\text{rgb}}$ does *not* contain `e(red,red)`, `e(green,green)`, `e(blue,blue)`.

Conversely, we claim that if the graph of Fig. 4 is 3-colorable, then there is a valuation mapping $\mathcal{B}_{\text{graph}}$ into $\mathcal{B}_{\text{rgb}}$. The proof of this claim is straightforward and left as an exercise.

To conclude, asking whether a graph is 3-colorable is the same as asking whether q_{rgb} is contained in the query encoding that graph.

Finally, note that by Proposition 2, $q_{\text{rgb}} \sqsubseteq q_{\text{graph}}$ is equivalent to `coloring` $\in eval(q_{\text{graph}}, \mathcal{B}_{\text{rgb}})$. The latter condition can be tested by the following DLV program.

```
%%% Query q_graph
coloring :- e(A,B), e(B,C), e(C,D), e(D,E), e(E,F), e(F,G), e(G,H), e(H,A), e(D,H),
            e(B,F), e(I,C), e(C,J), e(J,G), e(G,I), e(A,I), e(B,H), e(E,J), e(A,E).
%%% Body of q_rgb
e(red,green). e(green,blue). e(blue,red).
e(green,red). e(blue,green). e(red,blue).
```

When this DLV program is executed, it turns out that `coloring` is not in the answer. Therefore, it is correct to conclude that the graph of Fig. 4 is *not* 3-colorable.

Exercise 4.1. The query q_{graph} encodes every undirected edge $\{\mathtt{X}, \mathtt{Y}\}$ as either `e(X,Y)` or `e(Y,X)` (but not both). Show that this encoding is correct for our purpose, and that there is no need to encode an edge as both `e(X,Y)` and `e(Y,X)`. □

Exercise 4.2. We first recall some notions from graph theory, which are useful for this exercise. An *undirected graph* is a pair $G = (V, E)$ where V is a finite set of *vertices* and E is a set of pairs of distinct vertices. The pairs in E are called *edges*. An *endomorphism* of G is a total function $f : V \to V$ such

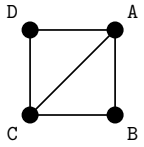

Figure 5: Undirected graph.

that for every edge $\{s,t\}$ in E, we have that $\{f(s),f(t)\}$ is also an edge in E. Obviously, if f maps two distinct vertices to a same vertex, then f is not bijective and $\{f(s) \mid s \in V\}$ is a strict subset of V.

For $V' \subseteq V$, the *subgraph* of G induced by V' is the undirected graph whose set of vertices is V', and whose edges are all those edges of E that are included in V'. For example, for $G = (V,E)$ with $V = \{a,b,c,d\}$ and $E = \{\{a,b\},\{b,c\},\{c,d\},\{d,a\},\{a,c\}\}$, the subgraph induced by $\{b,c,d\}$ has edges $\{b,c\}$ and $\{c,d\}$ (and has no other edges).

A *core* of G is a smallest subgraph of G for which every endomorphism is a bijective.

Consider now the conjunctive query:

```
c :- e(A,B), e(B,A), e(B,C), e(C,B), e(C,D), e(D,C), e(A,D), e(D,A), e(A,C), e(C,A).
```

The body of this query encodes the undirected graph of Fig. 5. Note that for every undirected edge $\{\texttt{X},\texttt{Y}\}$, the body contains both `e(X,Y)` and `e(Y,X)`. Compute (by hand and/or by relying on DLV) a minimized query that is equivalent to the above query. Can you give a graph-theoretical interpretation of this minimized query? Would that interpretation still be correct if edges $\{\texttt{X},\texttt{Y}\}$ would be encoded as either `e(X,Y)` or `e(Y,X)` (but not both). □

4.2 Data and Query Complexity

In *A Datalog Primer*, it was argued that every stratified Datalog program executes in polynomial time. Every conjunctive query is a Datalog program, and can thus be executed in polynomial time (and even in logarithmic space—see the document entitled *Adding Recursion to SPJRUD*). That is, for every fixed conjunctive query q, the following problem can be solved in polynomial time:

Data Complexity.

INPUT: A database instance I.

QUESTION: Compute $eval(q,I)$.

On the other hand, 3-COLORABILITY is known to be an NP-complete problem, which means that if P ≠ NP (which is mostly believed), then 3-COLORABILITY cannot be solved in polynomial time.

It is correct to conclude that if P ≠ NP, then 3-COLORABILITY cannot be solved by a conjunctive query.

But wait a minute... Haven't we showed in Section 4.1 that 3-COLORABILITY can be solved by a conjunctive query? No, we haven't... What we showed in that section is that for the fixed database instance defined as $\mathcal{B}_{rgb}$:= {`e(red,green)`, `e(green,blue)`, `e(blue,red)`, `e(green,red)`, `e(blue,green)`, `e(red,blue)`}, the following problem cannot be solved in polynomial time (under the assumption P ≠ NP):

Query Complexity.

INPUT: A conjunctive query q with head `coloring`.

QUESTION: Is `coloring` $\in eval(q,\mathcal{B}_{rgb})$?

In the database literature, we distinguish between *data complexity* and *query complexity* of query evaluation, depending on whether the database instance or the query is considered as the input. In database

systems, data complexity is the more significant complexity measure, because database instances are typically very much larger than queries. Note that our query q_{graph} is atypical in this respect, because its body is an undirected graph, which can be very large. The analysis in the current section shows that the query complexity for conjunctive queries is higher than the data complexity: NP-complete versus logspace.

5 Unions of Conjunctive Queries

In this section, we show that the results for conjunctive queries proved so far can be easily extended to incorporate also union (or disjunction). However, these results fail if we also add negation or recursion.

5.1 The Containment Problem for UCQs

A *Union of Conjunctive Queries (UCQ)* is a finite set of conjunctive queries, all with the same head predicate. For example, the following UCQ consists of three conjunctive queries:

```
happy(X) :- owns(X,iPad)
happy(Person) :- owns(Person,iPod)
happy(X) :- knows(X,Person), owns(Person,iPad), owns(Person,iPod)
```

Formally, a UCQ Q is a finite set $\{q_1, q_2, \ldots, q_n\}$ of conjunctive queries, all with the same head predicate. The answer to Q on a database I, denoted $eval(Q, I)$, is equal to $eval(q_1, I) \cup eval(q_2, I) \cup \cdots \cup eval(q_n, I)$. Informally, every conjunctive query is executed, and the answers are grouped together.

The CONTAINMENT PROBLEM for conjunctive queries and UCQs is the following problem:

INPUT: A conjunctive query q, a UCQ $Q = \{q_1, q_2, \ldots, q_n\}$.

QUESTION: Is it the case that $q \sqsubseteq Q$, i.e., do we have $eval(q, I) \subseteq eval(Q, I)$ for every database instance I?

We will show the following result.

Proposition 3. *Let q be a conjunctive query and $Q = \{q_1, q_2, \ldots, q_n\}$ a UCQ.*
Then, $q \sqsubseteq Q$ if and only if for some $i \in \{1, \ldots, n\}$, we have $q \sqsubseteq q_i$.

The implication $\Longleftarrow$ in Proposition 3 is trivial. On the other hand, the direction $\Longrightarrow$ may be surprising at first sight. Note, for example, that in set theory, $A \subseteq A_1 \cup A_2 \cup \cdots \cup A_n$ does not imply that A is contained in some A_i. For example, for $A = \{1, 2\}$, $A_1 = \{1\}$, and $A_2 = \{2\}$, we have that $A \subseteq A_1 \cup A_2$, but $A \not\subseteq A_1$ and $A \not\subseteq A_2$.

Like the earlier containment results for conjunctive queries, the proof of $\Longrightarrow$ in Proposition 3 uses that canonical bodies *are* database instances. To prove this direction, assume $q \sqsubseteq Q$. Let q be the query $q : \mathcal{H} :- \mathcal{B}$, and let $\mathfrak{h} :- \mathfrak{b}$ be its canonical ground rule. Clearly, $\mathfrak{h} \in eval(q, \mathfrak{b})$. By our hypothesis that $q \sqsubseteq Q$, it follows $\mathfrak{h} \in eval(Q, \mathfrak{b})$. Since $eval(Q, \mathfrak{b}) = eval(q_1, \mathfrak{b}) \cup eval(q_2, \mathfrak{b}) \cup \cdots \cup eval(q_n, \mathfrak{b})$, it follows that there must be $i \in \{1, \ldots, n\}$ such that $\mathfrak{h} \in eval(q_i, \mathfrak{b})$. Then, by Proposition 2, $q \sqsubseteq q_i$, which concludes the proof.

The CONTAINMENT PROBLEM for UCQs is the following problem:

INPUT: Two UCQs $P = \{p_1, p_2, \ldots, p_m\}$ and $Q = \{q_1, q_2, \ldots, q_n\}$.

QUESTION: Is it the case that $P \sqsubseteq Q$, i.e., do we have $eval(P, I) \subseteq eval(Q, I)$ for every database instance I?

Proposition 4. *Let P and Q be as previously stated, Then, $P \sqsubseteq Q$ if and only if for every $i \in \{1, \ldots, m\}$, there exists $j \in \{1, \ldots, n\}$ such that $p_i \sqsubseteq q_j$.*

Exercise 5.1. Prove Proposition 4. *Hint:* The proof is a simple extension of the proof of Proposition 3. □

Exercise 5.2. Show that the language of UCQs has the same expressive power as SPJRU, i.e., the relational algebra without difference. To this end, show that each of the following equivalences is correct:

$$\begin{aligned}
\sigma_{A=c}(E \cup F) &\equiv \sigma_{A=c}(E) \cup \sigma_{A=c}(F) \\
\sigma_{A=B}(E \cup F) &\equiv \sigma_{A=B}(E) \cup \sigma_{A=B}(F) \\
\pi_X(E \cup F) &\equiv \pi_X(E) \cup \pi_X(F) \\
\rho_{A \mapsto B}(E \cup F) &\equiv \rho_{A \mapsto B}(E) \cup \rho_{A \mapsto B}(F) \\
E \bowtie (F \cup G) &\equiv (E \bowtie F) \cup (E \bowtie G) \\
(E \cup F) \bowtie G &\equiv (E \bowtie G) \cup (F \bowtie G)
\end{aligned}$$

Then show the following: repeated application of these equivalences (from left to right) rewrites an SPJRU expression E into an equivalent expression of the form $E_1 \cup E_2 \cup \cdots \cup E_\ell$ where each E_i is union-free (i.e., each E_i is a conjunctive query). Note that the last two rules result in an exponential blowup in the size of the query (but that does not matter if we are only concerned about data complexity). □

5.2 Minimization of UCQs

QUERY MINIMIZATION for UCQs is the following task:

INPUT: A UCQ $Q = \{q_1, \ldots, q_n\}$.

QUESTION: Find a smallest UCQ equivalent to Q.

Now, the minimization consists in two tasks:

1. if $q_i \sqsubseteq q_j$ with $i \neq j$, then remove q_i; and
2. minimize all rule bodies.

For example, in the following UCQ, the last rule is redundant and can therefore be removed.

```
happy(X) :- owns(X,iPad)
happy(X) :- owns(X,iPod)
happy(X) :- owns(X,iPad), owns(X,iPod)
```

Exercise 5.3. Minimize the following UCQ.

```
answer(X) :- r(Y,X), r(X,Y), r(X,Z), r(Z,X)
answer(Z) :- r(X,Y), r(Y,Z), r(Z,V), r(V,X)
```

Show that the result of Section 3.2 extends to UCQs: for every UCQ, there is a unique (up to variable renaming) equivalent UCQ that has a minimal total number of atoms. □

5.3 Beyond UCQ

Proposition 4 implies that there exists an algorithm for the CONTAINMENT PROBLEM for UCQs, and therefore we can also effectively check whether two UCQs are equivalent.

The CONTAINMENT PROBLEM for languages $\mathcal{L}_1$ and $\mathcal{L}_2$ is the following problem:

INPUT: A query $q_1 \in \mathcal{L}_1$ and a query $q_2 \in \mathcal{L}_2$.

QUESTION: Is $q_1 \sqsubseteq q_2$?

In many studies, it will be the case that $\mathcal{L}_1 = \mathcal{L}_2$, in which case we speak about the CONTAINMENT PROBLEM for the language $\mathcal{L}_1$. It is known that there exists no algorithm for the CONTAINMENT PROBLEM for relational algebra. Likewise, there exists no algorithm for the CONTAINMENT PROBLEM for positive Datalog with recursion. By positive Datalog, we mean Datalog without negation. Thus, the CONTAINMENT PROBLEM for UCQs is decidable, but becomes undecidable as soon as we add negation or recursion.

A A Note on the Complexity of Query Minimization

Assume that we are given an undirected graph, for example, the graph of Fig. 4. Now consider the following two conjunctive queries, which we call q_{long} and q_{short} respectively:

```
%%% query q_long
coloring :- e(A,B), e(B,C), e(C,D), e(D,E), e(E,F), e(F,G), e(G,H), e(H,A), e(D,H),
            e(B,F), e(I,C), e(C,J), e(J,G), e(G,I), e(A,I), e(B,H), e(E,J), e(A,E),
            e(red,green), e(green,blue), e(blue,red),
            e(green,red), e(blue,green), e(red,blue).
```

and

```
%%% query q_short
coloring :- e(red,green), e(green,blue), e(blue,red),
            e(green,red), e(blue,green), e(red,blue).
```

Note that q_{short} is a query without variables, which is strange, but not forbidden. It is easily verified that q_{short} is a minimal query.

Now let us ask the question:

> Is q_{long} equivalent to q_{short}? In other words, if we solve the problem QUERY MINIMIZATION with q_{long} as input, do we find q_{short} as output?

Reasoning along the lines of Section 4.1, it can easily be verified that the answer to this question is "yes" if the graph of Fig. 4 is 3-colorable; and "no" if that graph is not 3-colorable.

The foregoing shows that an algorithm for QUERY MINIMIZATION can be used to solve 3-COLORABILITY. Now it is known that if P $\neq$ NP, then 3-COLORABILITY cannot be solved by a polynomial-time algorithm. Consequently, if P $\neq$ NP, then there is no polynomial-time algorithm for QUERY MINIMIZATION.