# 4 Conjunctive Queries

**Alice:** *Shall we start asking queries?*
**Sergio:** *Very simple ones for the time being.*
**Riccardo:** *But the system will answer them fast.*
**Vittorio:** *And there is some nice theory.*

In this chapter we embark on the study of queries for relational databases, a rich topic that spans a good part of this book. This chapter focuses on a limited but extremely natural and commonly arising class of queries called *conjunctive queries*. Five equivalent versions of this query family are presented here: one from each of the calculus and datalog paradigms, two from the algebra paradigm, and a final one that has a more visual form. In the context of conjunctive queries, the three nonalgebraic versions can be viewed as minor syntactic variants of each other; but these similarities diminish as the languages are generalized to incorporate negation and/or recursion. This chapter also discusses query composition and its interaction with user views, and it extends conjunctive queries in a straightforward manner to incorporate union (or disjunction).

The conjunctive queries enjoy several desirable properties, including, for example, decidability of equivalence and containment. These results will be presented in Chapter 6, in which a basic tool, the Homomorphism Theorem, is developed. Most of these results extend to conjunctive queries with union.

In the formal framework that we have developed in this book, we distinguish between a *query*, which is a syntactic object, and a *query mapping*, which is the function defined by a query interpreted under a specified semantics. However, we often blur these two concepts when the meaning is clear from the context. In the relational model, query mappings generally have as domain the family of all instances of a specified relation or database schema, called the *input schema*; and they have as range the family of instances of an *output schema*, which might be a database schema or a relation schema. In the latter case, the relation name may be specified as part of the syntax of the query or by the context, or it may be irrelevant to the discussion and thus not specified at all. We generally say that a query (mapping) is *from* (or *over*) its input schema *to* its output schema. Finally, two queries $q_1$ and $q_2$ over $\mathbf{R}$ are *equivalent*, denoted $q_1 \equiv q_2$, if they have the same output schema and $q_1(\mathbf{I}) = q_2(\mathbf{I})$ for each instance $\mathbf{I}$ over $\mathbf{R}$.

This chapter begins with an informal discussion that introduces a family of simple queries and illustrates one approach to expressing them formally. Three versions of conjunctive queries are then introduced, and all of them have a basis in logic. Then a brief

**(4.1)** Who is the director of "Cries and Whispers"?
**(4.2)** Which theaters feature "Cries and Whispers"?
**(4.3)** What are the address and phone number of the Le Champo?
**(4.4)** List the names and addresses of theaters featuring a Bergman film.
**(4.5)** Is a film directed by Bergman playing in Paris?
**(4.6)** List the pairs of persons such that the first directed the second in a movie, and vice versa.
**(4.7)** List the names of directors who have acted in a movie they directed.
**(4.8)** List pairs of actors that acted in the same movie.
**(4.9)** On any input produce ⟨"Apocalypse Now", "Coppola"⟩ as the answer.
**(4.10)** Where can I see "Annie Hall" or "Manhattan"?
**(4.11)** What are the films with Allen as actor or director?
**(4.12)** What films with Allen as actor or director are currently featured at the Concorde?
**(4.13)** List all movies that were directed by Hitchcock or that are currently playing at the Rex.
**(4.14)** List all actors and director of the movie "Apocalypse Now."

**Figure 4.1:** Examples of conjunctive queries, some of which require union

digression is made to consider query composition and database views. The algebraic perspectives on conjunctive queries are then given, along with the theorem showing the equivalence of all five approaches to conjunctive queries. Finally, various forms of union and disjunction are added to the conjunctive queries.

## 4.1 Getting Started

To present the intuition of conjunctive queries, consider again the **CINEMA** database of Chapter 3. The following correspond to conjunctive queries:

**(4.1)** Who is the director of "Cries and Whispers"?
**(4.2)** Which theaters feature "Cries and Whispers"?
**(4.3)** What are the address and phone number of the Le Champo?

These and other queries used in this section are gathered in Fig. 4.1. Each of the queries just given calls for extracting information from a single relation. In contrast, queries (4.4) through (4.7) involve more than one relation.

In queries (4.1–4.4 and 4.6–4.9), the database is asked to find values or tuples of values for which a certain pattern of data holds in the database, and in query (4.5) the database is asked whether a certain pattern of data holds. We shall see that the patterns can be described simply in terms of the existence of tuples that are connected to each other by equality of some of their coordinates. On the other hand, queries (4.10) through (4.14) cannot be expressed in this manner unless some form of disjunction or union is incorporated.

**EXAMPLE 4.1.1** Consider query (4.4). Intuitively, we express this query by stating that

**if** there are tuples $r_1, r_2, r_3$ respectively in relations
*Movies*, *Pariscope*, *Location* **such that**
the *Director* in $r_1$ is "Bergman"
**and** the *Titles* in tuple $r_1$ and $r_2$ are the same
**and** the *Theaters* in tuple $r_2$ and $r_3$ are the same
**then** we want the *Theater* and *Address* coordinates from tuple $r_3$.

In this formulation we essentially use variables that range over tuples. Although this is the basis of the so-called (*relational*) *tuple calculus* (see Exercise 5.23 in the next chapter), the focus of most theoretical investigations has been on the *domain calculus*, which uses variables that range over constants rather than tuples. This also reflects the convention followed in the predicate calculus of first-order logic. Thus we reformulate the preceding query as

**if** there are tuples $\langle x_{ti}$, "Bergman", $x_{ac}\rangle$, $\langle x_{th}, x_{ti}, x_s\rangle$, and $\langle x_{th}, x_{ad}, x_p\rangle$,
respectively, in relations *Movies, Pariscope,* and *Location*
**then** include the tuple $\langle Theater : x_{th},\ Address : x_{ad}\rangle$ in the answer,

where $x_{ti}, x_{ac}, \ldots$ are variables. Note that the equalities specified in the first formulation are achieved implicitly in the second formulation through multiple occurrences of variables.

The translation of this into the syntax of *rule-based conjunctive queries* is now obtained by

$$ans(x_{th}, x_{ad}) \leftarrow Movies(x_{ti}, \text{"Bergman"}, x_{ac}),\ Pariscope(x_{th}, x_{ti}, x_s),\ Location(x_{th}, x_{ad}, x_p)$$

where *ans* (for "answer") is a relation over {*Theater*, *Address*}. The atom to the left of the $\leftarrow$ is called the rule *head*, and the set of atoms to the right is called the *body*.

The preceding rule may be abbreviated as

$$ans(x_{th}, x_{ad}) \leftarrow Movies(x_{ti}, \text{"Bergman"}, \_),\ Pariscope(x_{th}, x_{ti}, \_),\ Location(x_{th}, x_{ad}, \_)$$

where _ is used to replace all variables that occur exactly once in the rule. Such variables are sometimes called *anonymous*.

In general, a rule-based conjunctive query is a single rule that has the form illustrated in the preceding example. The semantics associated with rule-based conjunctive queries ensures that their interpretation corresponds to the more informal expressions given in the preceding example. Rule-based conjunctive queries can be viewed as the basic building block for datalog, a query language based on logic programming that provides an elegant syntax for expressing recursion.

A second paradigm for the conjunctive queries has a more visual form and uses tables with variables and constants. Although we present a more succinct formalism for this

| *Movies* | *Title* | *Director* | *Actor* |
|---|---|---|---|
| | _The Seventh Seal | Bergman | |

| *Pariscope* | *Theater* | *Title* | *Schedule* |
|---|---|---|---|
| | _Rex | _The Seventh Seal | |

| *Location* | *Theater* | *Address* | *Phone number* |
|---|---|---|---|
| | P._Rex | P._1 bd. Poissonnière | |

**Figure 4.2:** A query in QBE

paradigm later in this chapter, we illustrate it in Fig. 4.2 with a query presented in the syntax of the language Query-By-Example (QBE) (see also Chapter 7). The identifiers starting with a _ designate variables, and P. indicates what to output. Following the convention established for QBE, variable names are chosen to reflect typical values that they might take. Note that the coordinate entries left blank correspond, in terms of the rule given previously, to distinct variables that occur exactly once in the body and do not occur in the head (i.e., to anonymous variables).

The third version of conjunctive queries studied in this chapter is a restriction of the predicate calculus; as will be seen, the term *conjunctive query* stems from this version. The fourth and fifth versions are algebraic in nature, one for the unnamed perspective and the other for the named perspective.

## 4.2 Logic-Based Perspectives

In this section we introduce and study three versions of the conjunctive queries, all stemming from mathematical logic. After showing the equivalence of the three resulting query languages, we extend them by incorporating a capability to express equality explicity, thereby yielding a slightly more powerful family of languages.

### Rule-Based Conjunctive Queries

The rule-based version of conjunctive queries is now presented formally. As will be seen later, the rule-based paradigm is well suited for specifying queries from database schemas to database schemas. However, to facilitate the comparison between the different variants of the conjunctive queries, we focus first on rule-based queries whose targets are relation schemas. We adopt the convention of using the name *ans* to refer to the name of the target relation if the name itself is unimportant (as is often the case with relational queries).

**Definition 4.2.1** Let $\mathbf{R}$ be a database schema. A *rule-based conjunctive query* over $\mathbf{R}$ is an expression of the form

$$ans(u) \leftarrow R_1(u_1), \ldots, R_n(u_n)$$

where $n \geq 0$, $R_1, \ldots, R_n$ are relation names in $\mathbf{R}$; *ans* is a relation name not in $\mathbf{R}$; and $u, u_1, \ldots, u_n$ are free tuples (i.e., may use either variables or constants). Recall that if $v = \langle x_1, \ldots, x_m \rangle$, then '$R(v)$' is a shorthand for '$R(x_1, \ldots, x_m)$'. In addition, the tuples $u, u_1, \ldots, u_n$ must have the appropriate arities (i.e., $u$ must have arity of *ans*, and $u_i$ must have the arity of $R_i$ for each $i \in [1, n]$). Finally, each variable occurring in $u$ must also occur at least once in $u_1, \ldots, u_n$. The set of variables occurring in $q$ is denoted $var(q)$.

Rule-based conjunctive queries are often more simply called *rules*. In the preceding rule, the subexpression $R_1(u_1), \ldots, R_n(u_n)$ is the *body* of the rule, and '$ans(u)$' is the *head*. The rule here is required by the definition to be *range restricted* (i.e., each variable occurring in the head must also occur in the body). Although this restriction is followed in most of the languages based on the use of rules, it will be relaxed in Chapter 18.

Intuitively, a rule may be thought of as a tool for deducing new facts. If one can find values for the variables of the rule such that the body holds, then one may deduce the head fact. This concept of "values for the variables in the rules" is captured by the notion of "valuation." Formally, given a finite subset $V$ of **var**, a *valuation* $\nu$ over $V$ is a total function $\nu$ from $V$ to the set **dom** of constants. This is extended to be identity on **dom** and then extended to map free tuples to tuples in the natural fashion.

We now define the semantics for rule-based conjunctive queries. Let $q$ be the query given earlier, and let $\mathbf{I}$ be an instance of $\mathbf{R}$. The *image* of $\mathbf{I}$ under $q$ is

$$q(\mathbf{I}) = \{\nu(u) \mid \nu \text{ is a valuation over } var(q) \text{ and } \nu(u_i) \in \mathbf{I}(R_i), \text{ for each } i \in [1, n]\}.$$

The *active domain* of a database instance $\mathbf{I}$, denoted $adom(\mathbf{I})$, is the set of all constants occurring in $\mathbf{I}$, and the active domain $adom(I)$ of relation instance $I$ is defined analogously. In addition, the set of constants occurring in a query $q$ is denoted $adom(q)$. We use $adom(q, \mathbf{I})$ as an abbreviation for $adom(q) \cup adom(\mathbf{I})$.

Let $q$ be a rule and $\mathbf{I}$ an input instance for $q$. Because $q$ is range restricted, it is easily verified that $adom(q(\mathbf{I})) \subseteq adom(q, \mathbf{I})$ (see Exercise 4.2). In other words, $q(\mathbf{I})$ contains only constants occurring in $q$ or in $\mathbf{I}$. In particular, $q(\mathbf{I})$ is finite, and so it is an instance.

A straightforward algorithm for evaluating a rule $q$ is to consider systematically all valuations with domain the set of variables occurring in $q$, and range the set of all constants occurring in the input or $q$. More efficient algorithms may be achieved, both by performing symbolic manipulations of the query and by using auxiliary data structures such as indexes. Such improvements are considered in Chapter 6.

Returning to the intuition, under the usual perspective a fundamental difference between the head and body of a rule $R_0 \leftarrow R_1, \ldots, R_n$ is that body relations are viewed as being stored, whereas the head relation is not. Thus, referring to the rule given earlier, the values of relations $R_1, \ldots, R_n$ are known because they are provided by the input instance

**I**. In other words, we are given the extension of $R_1, \ldots, R_n$; for this reason they are called *extensional* relations. In contrast, relation $R_0$ is not stored and its value is computed on request by the query; the rule gives only the "intension" or definition of $R_0$. For this reason we refer to $R_0$ as an *intensional* relation. In some cases, the database instance associated with $R_1, \ldots, R_n$ is called the *extensional database* (edb), and the rule itself is referred to as the *intensional database* (idb). Also, the defined relation is sometimes referred to as an *idb relation*.

We now present the first theoretical property of conjunctive queries. A query $q$ over $\mathbf{R}$ is *monotonic* if for each $\mathbf{I}$, $\mathbf{J}$ over $\mathbf{R}$, $\mathbf{I} \subseteq \mathbf{J}$ implies that $q(\mathbf{I}) \subseteq q(\mathbf{J})$. A query $q$ is *satisfiable* if there is some input $\mathbf{I}$ such that $q(\mathbf{I})$ is nonempty.

**PROPOSITION 4.2.2** Conjunctive queries are monotonic and satisfiable.

*Proof* Let $q$ be the rule-based conjunctive query

$$ans(u) \leftarrow R_1(u_1), \ldots, R_n(u_n).$$

For monotonicity, let $\mathbf{I} \subseteq \mathbf{J}$, and suppose that $t \in q(\mathbf{I})$. Then for some valuation $\nu$ over $var(q)$, $\nu(u_i) \in \mathbf{I}(R_i)$ for each $i \in [1, n]$, and $t = \nu(u)$. Because $\mathbf{I} \subseteq \mathbf{J}$, $\nu(u_i) \in \mathbf{J}(R_i)$ for each $i$, and so $t \in q(\mathbf{J})$.

For satisfiability, let $\mathbf{d}$ be the set of constants occurring in $q$, and let $a \in \mathbf{dom}$ be new. Define $\mathbf{I}$ over the relation schemas $R$ of the rule body so that

$$\mathbf{I}(R) = (\mathbf{d} \cup \{a\})^{arity(R)}$$

[i.e., the set of all tuples formed from $(\mathbf{d} \cup \{a\})$ having arity $arity(R)$]. Finally, let $\nu$ map each variable in $q$ to $a$. Then $\nu(u_i) \in \mathbf{I}(R_i)$ for $i \in [1, n]$, and so $\nu(u) \in q(\mathbf{I})$. Thus $q$ is satisfiable. ■

The monotonicity of the conjunctive queries points to limitations in their expressive power. Indeed, one can easily exhibit queries that are nonmonotonic and therefore not conjunctive queries. For instance, the query "Which theaters in New York show only Woody Allen films?" is nonmonotonic.

We close this subsection by indicating how rule-based conjunctive queries can be used to express yes-no queries. For example, consider the query

**(4.5)** Is there a film directed by Bergman playing in Paris?

To provide an answer, we assume that relation name *ans* has arity 0. Then applying the rule

$$ans() \leftarrow Movies(x, \text{"Bergman"}, y), Pariscope(z, x, w)$$

returns the relation $\{\langle\rangle\}$ if the answer is yes, and returns $\{\}$ if the answer is no.

### Tableau Queries

If we blur the difference between a variable and a constant, the body of a conjunctive query can be seen as an instance. This leads to a formulation of conjunctive queries called "tableau", which is closest to the visual form provided by QBE.

**DEFINITION 4.2.3** The notion of *tableau* over a schema $\mathbf{R}$ ($R$) is defined exactly as was the notion of instance over $\mathbf{R}$ ($R$), except that both variables and constants may occur. A *tableau query* is simply a pair $(\mathbf{T}, u)$ [or $(T, u)$] where $\mathbf{T}$ is a tableau and each variable in $u$ also occurs in $\mathbf{T}$. The free tuple $u$ is called the *summary* of the tableau query.

The summary tuple $u$ in a tableau query $(\mathbf{T}, u)$ represents the tuples included in the answer to the query. Thus the answer consists of all tuples $u$ for which the pattern described by $\mathbf{T}$ is found in the database.

**EXAMPLE 4.2.4** Let $\mathbf{T}$ be the tableau

| *Movies* | *Title* | *Director* | *Actor* |
|---|---|---|---|
| | $x_{ti}$ | "Bergman" | $x_{ac}$ |

| *Pariscope* | *Theater* | *Title* | *Schedule* |
|---|---|---|---|
| | $x_{th}$ | $x_{ti}$ | $x_s$ |

| *Location* | *Theater* | *Address* | *Phone Number* |
|---|---|---|---|
| | $x_{th}$ | $x_{ad}$ | $x_p$ |

The tableau query $(\mathbf{T}, \langle \mathit{Theater} : x_{th}, \ \mathit{Address} : x_{ad} \rangle)$ expresses query (4.4). If the unnamed perspective on tuples is used, then the names of the attributes are not included in $u$.

The notion of valuation is extended in the natural fashion to map tableaux[1] to instances. An *embedding* of tableau $\mathbf{T}$ into instance $\mathbf{I}$ is a valuation $\nu$ for the variables occurring in $\mathbf{T}$ such that $\nu(\mathbf{T}) \subseteq \mathbf{I}$. The semantics for tableau queries is essentially the same as for rule-based conjunctive queries: The output of $(\mathbf{T}, u)$ on input $\mathbf{I}$ consists of all tuples $\nu(u)$ where $\nu$ is an embedding of $\mathbf{T}$ into $\mathbf{I}$.

Aside from the fact that tableau queries do not indicate a relation name for the answer, they are syntactically close to the rule-based conjunctive queries. Furthermore, the alternative perspective provided by tableaux lends itself to the development of several natural results. Perhaps the most compelling of these arises in the context of the chase (see

[1] One *tableau*, two *tableaux*.

Chapter 8), which provides an elegant characterization of two conjunctive queries yielding identical results when the inputs satisfy certain dependencies.

A family of restricted tableaux called *typed* have been used to develop a number of theoretical results. A tableau query $q = (T, u)$ under the named perspective, where $T$ is over relation schema $R$ and $sort(u) \subseteq sort(R)$, is typed if no variable of $T$ or $t$ is associated with two distinct attributes in $q$. Intuitively, the term 'typed' is used because it is impossible for entries from different attributes to be compared. The connection between typed tableaux and conjunctive queries in the algebraic paradigm is examined in Exercises 4.19 and 4.20. Additional results concerning complexity issues around typed tableau queries are considered in Exercises 6.16 and 6.21 in Chapter 6. Typed tableaux also arise in connection with data dependencies, as studied in Part C.

### Conjunctive Calculus

The third formalism for expressing conjunctive queries stems from predicate calculus. (A review of predicate calculus is provided in Chapter 2, but the presentation of the calculus in this and the following chapter is self-contained.)

We begin by presenting conjunctive calculus queries that can be viewed as syntactic variants of rule-based conjunctive queries. They involve simple use of conjunction and existential quantification. As will be seen, the full conjunctive calculus, defined later, allows unrestricted use of conjunction and existential quantification. This provides more flexibility in the syntax but, as will be seen, does not increase expressive power.

Consider the conjunctive query

$$ans(e_1, \ldots, e_m) \leftarrow R_1(u_1), \ldots, R_n(u_n).$$

A conjunctive calculus query that has the same semantics is

$$\{e_1, \ldots, e_m \mid \exists x_1, \ldots, x_k(R_1(u_1) \wedge \cdots \wedge R_n(u_n))\},$$

where $x_1, \ldots, x_k$ are all the variables occurring in the body and not the head. The symbol $\wedge$ denotes conjunction (i.e., "and"), and $\exists$ denotes existential quantification (intuitively, $\exists x \ldots$ denotes "there exists an $x$ such that ..."). The term 'conjunctive query' stems from the presence of conjunctions in the syntax.

---

**Example 4.2.5** In the calculus paradigm, query (4.4) can be expressed as follows:

$$\{x_{th},\ x_{ad} \mid \exists x_{ti}\, \exists x_{ac}\, \exists x_s\, \exists x_p\, (Movies(x_{ti}, \text{“Bergman”}, x_{ac}) \wedge Pariscope(x_{th}, x_{ti}, x_s) \wedge Location(x_{th}, x_{ad}, x_p))\}.$$

Note that some but not all of the existentially quantified variables play the role of anonymous variables, in the sense mentioned in Example 4.1.1.

---

The syntax used here can be viewed as a hybrid of the usual set-theoretic notation,

used to indicate the form of the query output, and predicate calculus, used to indicate what should be included in the output. As discussed in Chapter 2, the semantics associated with calculus formulas is a restricted version of the conventional semantics found in first-order logic.

We now turn to the formal definition of the syntax and semantics of the (full) conjunctive calculus.

**DEFINITION 4.2.6** Let **R** be a database schema. A (*well-formed*) *formula* over **R** for the conjunctive calculus is an expression having one of the following forms:

(a) an atom over **R**;

(b) $(\varphi \wedge \psi)$, where $\varphi$ and $\psi$ are formulas over **R**; or

(c) $\exists x \varphi$, where $x$ is a variable and $\varphi$ is a formula over **R**.

In formulas we permit the abbreviation of $\exists x_1 \ldots \exists x_n$ by $\exists x_1, \ldots, x_n$.

The usual notion of "free" and "bound" occurrences of variables is now defined. An occurrence of variable $x$ in formula $\varphi$ is *free* if

(i) $\varphi$ is an atom; or

(ii) $\varphi = (\psi \wedge \xi)$ and the occurrence of $x$ is free in $\psi$ or $\xi$; or

(iii) $\varphi = \exists y \psi$, $x$ and $y$ are distinct variables, and the occurrence of $x$ is free in $\psi$.

An occurrence of $x$ in $\varphi$ is *bound* if it is not free. The set of *free variables* in $\varphi$, denoted $free(\varphi)$, is the set of all variables that have at least one free occurrence in $\varphi$.

**DEFINITION 4.2.7** A *conjunctive calculus query* over database schema **R** is an expression of the form

$$\{e_1, \ldots, e_m \mid \varphi\},$$

where $\varphi$ is a conjunctive calculus formula, $\langle e_1, \ldots, e_m \rangle$ is a free tuple, and the set of variables occurring in $\langle e_1, \ldots, e_m \rangle$ is exactly $free(\varphi)$. If the named perspective is being used, then attributes can be associated with output tuples by specifying a relation name $R$ of arity $m$. The notation

$$\{\langle e_1, \ldots, e_m \rangle : A_1 \ldots A_m \mid \varphi\}$$

can be used to indicate the sort of the output explicitly.

To define the semantics of conjunctive calculus queries, it is convenient to introduce some notation. Recall that for finite set $V \subset \mathbf{var}$, a *valuation* over $V$ is a total function $\nu$ from $V$ to **dom**. This valuation will sometimes be viewed as a syntactic expression of the form

$$\{x_1/a_1, \ldots, x_n/a_n\},$$

where $x_1, \ldots, x_n$ is a listing of $V$ and $a_i = \nu(x_i)$ for each $i \in [1, n]$. This may also be interpreted as a set. For example, if $x$ is not in the domain of $\nu$ and $c \in \mathbf{dom}$, then $\nu \cup \{x/c\}$ denotes the valuation with domain $V \cup \{x\}$ that is identical to $\nu$ on $V$ and maps $x$ to $c$.

Now let $\mathbf{R}$ be a database schema, $\varphi$ a conjunctive calculus formula over $\mathbf{R}$, and $\nu$ a valuation over $free(\varphi)$. Then $\mathbf{I}$ *satisfies* $\varphi$ under $\nu$, denoted $\mathbf{I} \models \varphi[\nu]$, if

(a) $\varphi = R(u)$ is an atom and $\nu(u) \in \mathbf{I}(R)$; or

(b) $\varphi = (\psi \wedge \xi)$ and[2] $\mathbf{I} \models \psi[\nu|_{free(\psi)}]$ and $\mathbf{I} \models \xi[\nu|_{free(\xi)}]$; or

(c) $\varphi = \exists x \psi$ and for some $c \in \mathbf{dom}$, $\mathbf{I} \models \psi[\nu \cup \{x/c\}]$.

Finally, let $q = \{e_1, \ldots, e_m \mid \varphi\}$ be a conjunctive calculus query over $\mathbf{R}$. For an instance $\mathbf{I}$ over $\mathbf{R}$, the *image* of $\mathbf{I}$ under $q$ is

$$q(\mathbf{I}) = \{\nu(\langle e_1, \ldots, e_n \rangle) \mid \mathbf{I} \models \varphi[\nu] \text{ and } \nu \text{ is a valuation over } free(\varphi)\}.$$

The *active domain* of a formula $\varphi$, denoted $adom(\varphi)$, is the set of constants occurring in $\varphi$; and as with queries $q$, we use $adom(\varphi, \mathbf{I})$ to abbreviate $adom(\varphi) \cup adom(\mathbf{I})$. An easy induction on conjunctive calculus formulas shows that if $\mathbf{I} \models \varphi[\nu]$, then the range of $\nu$ is contained in $adom(\mathbf{I})$ (see Exercise 4.3). This implies, in turn, that to evaluate a conjunctive calculus query, one need only consider valuations with range contained in $adom(\varphi, \mathbf{I})$ and, hence, only a finite number of them. This pleasant state of affairs will no longer hold when disjunction or negation is incorporated into the calculus (see Section 4.5 and Chapter 5).

Conjunctive calculus formulas $\varphi$ and $\psi$ over $\mathbf{R}$ are *equivalent* if they have the same free variables and, for each $\mathbf{I}$ over $\mathbf{R}$ and valuation $\nu$ over $free(\varphi) = free(\psi)$, $\mathbf{I} \models \varphi[\nu]$ iff $\mathbf{I} \models \psi[\nu]$. It is easily verified that if $\varphi$ and $\psi$ are equivalent, and if $\Psi'$ is the result of replacing an occurrence of $\varphi$ by $\psi$ in conjunctive calculus formula $\Psi$, then $\Psi$ and $\Psi'$ are equivalent (see Exercise 4.4).

It is easily verified that for all conjunctive calculus formulas $\varphi$, $\psi$, and $\xi$, $(\varphi \wedge \psi)$ is equivalent to $(\psi \wedge \varphi)$, and $(\varphi \wedge (\psi \wedge \xi))$ is equivalent to $((\varphi \wedge \psi) \wedge \xi)$. For this reason, we may view conjunction as a polyadic connective rather than just binary.

We next show that conjunctive calculus queries, which allow unrestricted nesting of $\exists$ and $\wedge$, are no more powerful than the simple conjunctive queries first exhibited, which correspond straightforwardly to rules. Thus the simpler conjunctive queries provide a normal form for the full conjunctive calculus. Formally, a conjunctive calculus query $q = \{u \mid \varphi\}$ is in *normal form* if $\varphi$ has the form

$$\exists x_1, \ldots, x_m (R_1(u_1) \wedge \cdots \wedge R_n(u_n)).$$

Consider now the two *rewrite* (or *transformation*) *rules* for conjunctive calculus queries:

*Variable substitution:* replace subformula

$$\exists x \ \psi \ \text{ by } \ \exists y \ \psi^x_y,$$

[2] $\nu|_V$ for variable set $V$ denotes the restriction of $\nu$ to $V$.

if $y$ does not occur in $\psi$, where $\psi_y^x$ denotes the formula obtained by replacing all free occurrences of $x$ by $y$ in $\psi$.

*Merge-exists:* replace subformula

$$(\exists y_1, \ldots, y_n \psi \wedge \exists z_1, \ldots, z_m \xi) \text{ by } \exists y_1, \ldots, y_n, z_1, \ldots, z_m (\psi \wedge \xi)$$

if $\{y_1, \ldots, y_n\}$ and $\{z_1, \ldots, z_m\}$ are disjoint, none of $\{y_1, \ldots, y_n\}$ occur (free or bound) in $\xi$, and none of $\{z_1, \ldots, z_m\}$ occur (free or bound) in $\psi$.

It is easily verified (see Exercise 4.4) that (1) application of these transformation rules to a conjunctive calculus formula yields an equivalent formula, and (2) these rules can be used to transform any conjunctive calculus formula into an equivalent formula in normal form. It follows that:

**Lemma 4.2.8** Each conjunctive calculus query is equivalent to a conjunctive calculus query in normal form.

We now introduce formal notation for comparing the expressive power of query languages. Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two query languages (with associated semantics). Then $\mathcal{Q}_1$ is *dominated* by $\mathcal{Q}_2$ (or, $\mathcal{Q}_1$ is *weaker than* $\mathcal{Q}_2$), denoted $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$, if for each query $q_1$ in $\mathcal{Q}_1$ there is a query $q_2$ in $\mathcal{Q}_2$ such that $q_1 \equiv q_2$. $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are *equivalent*, denoted $\mathcal{Q}_1 \equiv \mathcal{Q}_2$, if $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ and $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$.

Because of the close correspondence between rule-based conjunctive queries, tableau queries, and conjunctive calculus queries in normal form, the following is easily verified (see Exercise 4.15).

**Proposition 4.2.9** The rule-based conjunctive queries, the tableau queries, and the conjunctive calculus are equivalent.

Although straightforward, the preceding result is important because it is the first of many that show equivalence between the expressive power of different query languages. Some of these results will be surprising because of the high contrast between the languages.

### Incorporating Equality

We close this section by considering a simple variation of the conjunctive queries presented earlier, obtained by adding the capability of explicitly expressing equality between variables and/or constants. For example, query (4.4) can be expressed as

$$\begin{aligned} ans(x_{th}, x_{ad}) \leftarrow\ & Movies(x_{ti}, x_d, x_{ac}), x_d = \text{“Bergman”}, \\ & Pariscope(x_{th}, x_{ti}, x_s),\ Location(x_{th}, x_{ad}, x_p) \end{aligned}$$

and query (4.6) can be expressed as

$$ans(y_1, y_2) \leftarrow Movies(x_1, y_1, z_1),\ Movies(x_2, y_2, z_2),\ y_1 = z_2,\ y_2 = z_1.$$

It would appear that explicit equalities like the foregoing can be expressed by conjunctive queries without equalities by using multiple occurrences of the same variable or constant. Although this is basically true, two problems arise. First, unrestricted rules with equality may yield infinite answers. For example, in the rule

$$ans(x, y) \leftarrow R(x), y = z$$

$y$ and $z$ are not tied to relation $R$, and there are infinitely many valuations satisfying the body of the rule. To ensure finite answers, it is necessary to introduce an appropriate notion of range restriction. Informally, an unrestricted rule with equality is *range restricted* if the equalities require that each variable in the body be equal to some constant or some variable occurring in an atom $R(u_i)$; Exercise 4.5 explores the notion of range restriction in more detail. A *rule-based conjunctive query with equality* is a range-restricted rule with equality.

A second problem that arises is that the equalities in a rule with equality may cause the query to be unsatisfiable. (In contrast, recall that rules without equality are always satisfiable; see Proposition 4.2.2.) Consider the following query, in which $R$ is a unary relation and $a, b$ are distinct constants.

$$ans(x) \leftarrow R(x), x = a, x = b.$$

The equalities present in this query require that $a = b$, which is impossible. Thus there is no valuation satisfying the body of the rule, and the query yields the empty relation on all inputs. We use $q^{\emptyset:\mathbf{R},R}$ (or $q^{\emptyset}$ if $\mathbf{R}$ and $R$ are understood) to denote the query that maps all inputs over $\mathbf{R}$ to the empty relation over $R$. Finally, note that one can easily check if the equalities in a conjunctive query with equality are unsatisfiable (and hence if the query is equivalent to $q^{\emptyset}$). This is done by computing the transitive closure of the equalities in the query and checking that no two distinct constants are required to be equal. Each satisfiable rule with equality is equivalent to a rule without equality (see Exercise 4.5c).

One can incorporate equality into tableau queries in a similar manner by adding separately a set of required equalities. Once again, no expressive power is gained if the query is satisfiable. Incorporating equality into the conjunctive calculus is considered in Exercise 4.6.

## 4.3 Query Composition and Views

We now present a digression that introduces the important notion of query composition and describe its relationship to database views. A main result here is that the rule-based conjunctive queries with equality are closed under composition.

Consider a database $\mathbf{R} = \{R_1, \ldots, R_n\}$. Suppose that we have a query $q$ (in any of the preceding formalisms). Conceptually, this can be used to define a relation with new relation name $S_1$, which can be used in subsequent queries as any ordinary relation from $\mathbf{R}$. In particular, we can use $S_1$ in the definition of a new relation $S_2$, and so on. In this context, we could call each of $S_1, S_2, \ldots$ *intensional* (in contrast with the extensional relations of $\mathbf{R}$).

This perspective on query composition is expressed most conveniently within the rule-

based paradigm. Specifically, a *conjunctive query program* (with or without equality) is a sequence $P$ of rules having the form

$$\begin{aligned} S_1(u_1) &\leftarrow body_1 \\ S_2(u_2) &\leftarrow body_2 \\ &\vdots \\ S_m(u_m) &\leftarrow body_m, \end{aligned}$$

where each $S_i$ is distinct and not in $\mathbf{R}$; and for each $i \in [1, m]$, the only relation names that may occur in $body_i$ are $R_1, \ldots, R_n$ and $S_1, \ldots, S_{i-1}$. An instance $\mathbf{I}$ over $\mathbf{R}$ and the program $P$ can be viewed as defining values for all of $S_1, \ldots, S_m$ in the following way: For each $i \in [1, m]$, $[P(\mathbf{I})](S_i) = q_i([P(\mathbf{I})])$, where $q_i$ is the $i^{\text{th}}$ rule and defines relation $S_i$ in terms of $\mathbf{I}$ and the previous $S_j$'s. If $P$ is viewed as defining a single output relation, then this output is $[P(\mathbf{I})](S_m)$. Analogous to rule-based conjunctive queries, the relations in $\mathbf{R}$ are called *edb* relations, and the relations occurring in rule heads are called *idb* relations.

---

**Example 4.3.1** Let $\mathbf{R} = \{Q, R\}$ and consider the conjunctive query program

$$\begin{aligned} S_1(x, z) &\leftarrow Q(x, y), R(y, z, w) \\ S_2(x, y, z) &\leftarrow S_1(x, w), R(w, y, v), S_1(v, z) \\ S_3(x, z) &\leftarrow S_2(x, u, v), Q(v, z). \end{aligned}$$

Figure 4.3 shows an example instance $\mathbf{I}$ for $\mathbf{R}$ and the values that are associated to $S_1$, $S_2$, $S_3$ by $P(\mathbf{I})$.

It is easily verified that the effect of the first two rules of $P$ on $S_2$ is equivalent to the effect of the rule

$$\begin{aligned} S_2(x, y, z) \leftarrow Q(x_1, y_1), R(y_1, z_1, w_1), x = x_1, w = z_1, \\ R(w, y, v), Q(x_2, y_2), R(y_2, z_2, w_2), v = x_2, z = z_2. \end{aligned}$$

Alternatively, expressed without equality, it is equivalent to

$$S_2(x, y, z) \leftarrow Q(x, y_1), R(y_1, w, w_1), R(w, y, v), Q(v, y_2), R(y_2, z, w_2).$$

Note how variables are renamed to prevent undesired "cross-talk" between the different rule bodies that are combined to form this rule. The effect of $P$ on $S_3$ can also be expressed using a single rule without equality (see Exercise 4.7).

---

It is straightforward to verify that if a permutation $P'$ of $P$ (i.e., a listing of the elements of $P$ in a possibly different order) satisfies the restriction that relation names in a rule body must be in a previous rule head, then $P'$ will define the same mapping as $P$. This kind of consideration will arise in a richer context when stratified negation is considered in Chapter 15.

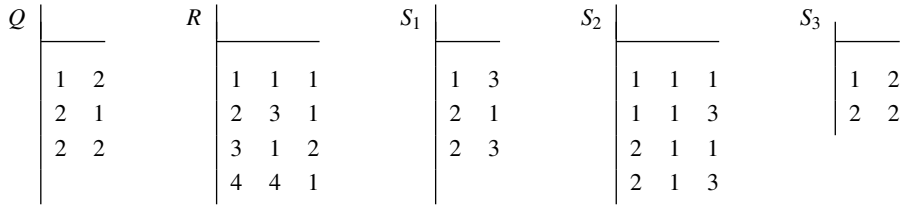


**Figure 4.3:** Application of a conjunctive query program

---

**EXAMPLE 4.3.2** Consider the following program $P$:

$$T(a, x) \leftarrow R(x)$$
$$S(x) \leftarrow T(b, x).$$

Clearly, $P$ always defines the empty relation $S$, so it is not equivalent to any rule-based conjunctive query without equality. Intuitively, the use of the constants $a$ and $b$ in $P$ masks the use of equalities, which in this case are contradictory and yield an unsatisfiable query.

---

Based on the previous examples, the following is easily verified (see Exercise 4.7).

**THEOREM 4.3.3 (Closure under Composition)** If conjunctive query program $P$ defines final relation $S$, then there is a conjunctive query $q$, possibly with equality, such that on all input instances $\mathbf{I}$, $q(\mathbf{I}) = [P(\mathbf{I})](S)$. Furthermore, if $P$ is satisfiable, then $q$ can be expressed without equality.

The notion of programs is based on the rule-based formalism of the conjunctive queries. In the other versions introduced previously and later in this chapter, the notation does not conveniently include a mechanism for specifying names for the output of intermediate queries. For the other formalisms we use a slightly more elaborate notation that permits the specification of these names. In particular, all of the formalisms are compatible with a functional, purely expression-based paradigm:

$$\begin{array}{lll} \textbf{let} & S_1 = q_1 & \textbf{in} \\ \textbf{let} & S_2 = q_2 & \textbf{in} \\ & \vdots & \\ \textbf{let} & S_{m-1} = q_{m-1} & \textbf{in} \\ & q_m & \end{array}$$

and with an imperative paradigm in which the intermediate query values are assigned to relation variables:

$$
\begin{aligned}
S_1 &:= q_1;\\
S_2 &:= q_2;\\
&\vdots\\
S_{m-1} &:= q_{m-1};\\
S_m &:= q_m.
\end{aligned}
$$

It is clear from Proposition 4.2.9 and Theorem 4.3.3 that the conjunctive calculus and tableau queries with equality are both closed under composition.

### Composition and User Views

Recall that the top level of the three-level architecture for databases (see Chapter 1) consists of user *views* (i.e., versions of the data that are restructured and possibly restricted images of the database as represented at the middle level). In many cases these views are specified as queries (or query programs). These may be *materialized* (i.e., a physical copy of the view is stored and maintained) or *virtual* (i.e., relevant information about the view is computed as needed). In the latter case, queries against the view generate composed queries against the underlying database, as illustrated by the following example.

---

**Example 4.3.4** Consider the view over schema {*Marilyn*, *Champo-info*} defined by the following two rules:

$$
\begin{aligned}
\mathit{Marilyn}(x_t) &\leftarrow \mathit{Movies}(x_t, x_d, \text{“Monroe”})\\
\mathit{Champo\text{-}info}(x_t, x_s, x_p) &\leftarrow \mathit{Pariscope}(\text{“Le Champo”}, x_t, x_s),\\
&\qquad\qquad \mathit{Location}(\text{“Le Champo”}, x_a, x_p).
\end{aligned}
$$

The conjunctive query “What titles in *Marilyn* are featured at the Le Champo at 21:00?” can be expressed against the view as

$$
\mathit{ans}(x_t) \leftarrow \mathit{Marilyn}(x_t),\ \mathit{Champo\text{-}info}(x_t, \text{“21:00”}, x_p).
$$

Assuming that the view is virtual, evaluation of this query is accomplished by considering the composition of the query with the view definition. This composition can be rewritten as

$$
\begin{aligned}
\mathit{ans}(x_t) \leftarrow\ & \mathit{Movies}(x_t, x_d, \text{“Monroe”}),\\
& \mathit{Pariscope}(\text{“Le Champo”}, x_t, \text{“21:00”})\\
& \mathit{Location}(\text{“Le Champo”}, x_a, x_p).
\end{aligned}
$$

An alternative expression specifying both view and query now follows. (Expressions from the algebraic versions of the conjunctive queries could also be used here.)

$$\begin{aligned} \mathit{Marilyn} &:= \{x_t \mid \exists x_d(\mathit{Movies}(x_t, x_d, \text{“Monroe”}))\}; \\ \mathit{Champo\text{-}info} &:= \{x_t, x_s, x_p \mid \exists x_a(\mathit{Location}(\text{“Le Champo”}, x_t, x_s) \\ &\qquad \wedge \mathit{Location}(\text{“Le Champo”}, x_a, x_p)\}; \\ \mathit{ans} &:= \{x_t \mid \mathit{Marilyn}(x_t) \wedge \exists x_p(\mathit{Champo\text{-}info}(x_t, \text{“21:00”}, x_p))\}. \end{aligned}$$

---

This example illustrates the case in which a query is evaluated over a single view; evaluation of the query involves a two-layer composition of queries. If a series of nested views is defined, then query evaluation can involve query compositions having two or more layers.

## 4.4 Algebraic Perspectives

The use of algebra operators provides a distinctly different perspective on the conjunctive queries. There are two distinct algebras associated with the conjunctive queries, and they stem, respectively, from the named, ordered-tuple perspective and the unnamed, function-based perspective. After presenting the two algebras, their equivalence with the conjunctive queries is discussed.

### The Unnamed Perspective: The SPC Algebra

The algebraic paradigm for relational queries is based on a family of unary and binary operators on relation instances. Although their application must satisfy some typing constraints, they are polymorphic in the sense that each of these operators can be applied to instances of an infinite number of arities or sorts. For example, as suggested in Chapter 3, the union operator can take as input any two relation instances having the same sort.

Three primitive algebra operators form the *unnamed conjunctive algebra*: selection, projection, and cross-product (or Cartesian product). This algebra is more often referred to as the *SPC algebra*, based on the first letters of the three operators that form it. (This convention will be used to specify other algebras as well.) An example is given before the formal definition of these operators.

---

**Example 4.4.1** We show how query (4.4) can be built up using the three primitive operators. First we use selection to extract the tuples of *Movies* that have Bergman as director.

$$I_1 := \sigma_{2=\text{“Bergman”}}(\mathit{Movies})$$

Next a family of “wide” (six columns wide, in fact) tuples is created by taking the cross-product of $I_1$ and *Pariscope*.

$$I_2 := I_1 \times \mathit{Pariscope}$$

Another selection is performed to focus on the members of $I_2$ that have first and fifth columns equal.

$$I_3 := \sigma_{1=5}(I_2)$$

In effect, the cross-product followed by this selection finds a matching of tuples from $I_1$ and *Pariscope* that agree on the *Title* coordinates.

At this point we are interested only in the theaters where these films are playing, so we use projection to discard the unneeded columns, yielding a unary relation.

$$I_4 := \pi_4(I_3)$$

Finally, this is paired with *Location* and projected on the *Theater* and *Address* columns to yield the answer.

$$I_5 := \pi_{2,3}(\sigma_{1=2}(I_4 \times \mathit{Location}))$$

The development just given uses SPC expressions in the context of a simple imperative language with assignment. In the pure SPC algebra, this query is expressed as

$$\pi_{2,3}(\sigma_{1=2}(\pi_4(\sigma_{1=5}(\sigma_{2=\text{“Bergman”}}(\mathit{Movies}) \times \mathit{Pariscope})) \times \mathit{Location})).$$

Another query that yields the same result is

$$\pi_{4,8}(\sigma_{4=7}(\sigma_{1=5}(\sigma_{2=\text{“Bergman”}}(\mathit{Movies} \times \mathit{Pariscope} \times \mathit{Location})))).$$

This corresponds closely to the conjunctive calculus query of Example 4.2.5.

---

Although the algebraic operators have a procedural feel to them, algebraic queries are used by most relational database systems as high-level specifications of desired output. Their actual implementation is usually quite different from the original form of the query, as will be discussed in Section 6.1.

We now formally define the three operators forming the SPC algebra.

*Selection:* This can be viewed as a “horizontal” operator. The two primitive forms are $\sigma_{j=a}$ and $\sigma_{j=k}$, where $j, k$ are positive integers and $a \in \mathbf{dom}$. [In practice, we usually surround constants with quotes (“ ”).] The operator $\sigma_{j=a}$ takes as input any relation instance $I$ with arity $\geq j$ and returns as output an instance of the same arity. In particular,

$$\sigma_{j=a}(I) = \{t \in I \mid t(j) = a\}.$$

The operator $\sigma_{j=k}$ for positive integers $j, k$ is defined analogously for inputs with arity $\geq \max\{j, k\}$. This is sometimes called *atomic* selection; generalizations of selection will be defined later.

*Projection:* This "vertical" operator can be used to delete and/or permute columns of a relation. The general form of this operator is $\pi_{j_1,\ldots,j_n}$, where $j_1, \ldots, j_n$ is a possibly empty sequence of positive integers (the empty sequence is written [ ]), possibly with repeats. This operator takes as input any relation instance with arity $\geq \max\{j_1, \ldots, j_n\}$ (where the max of $\emptyset$ is 0) and returns an instance with arity $n$. In particular,

$$\pi_{j_1,\ldots,j_n}(I) = \{\langle t(j_1), \ldots, t(j_n)\rangle \mid t \in I\}.$$

*Cross-product (or Cartesian product):* This operator provides the capability for combining relations. It takes as inputs a pair of relations having arbitrary arities $n$ and $m$ and returns a relation with arity $n + m$. In particular, if $arity(I) = n$ and $arity(J) = m$, then

$$I \times J = \{\langle t(1), \ldots, t(n), s(1), \ldots, s(m)\rangle \mid t \in I \text{ and } s \in J\}.$$

Cross-product is associative and noncommutative and has the nonempty 0-ary relation $\{\langle\rangle\}$ as left and right identity. Because it is associative, we sometimes view cross-product as a polyadic operator and write, for example, $I_1 \times \cdots \times I_n$.

We extend the cross-product operator to tuples in the natural fashion—that is $u \times v$ is a tuple with arity = $arity(u) + arity(v)$.

The SPC algebra is the family of well-formed expressions containing relation names and one-element unary constants and closed under the application of the selection, projection, and cross-product operators just defined. Each expression is considered to be defined over a given database schema and has an associated output arity. We now give the formal, inductive definition.

Let **R** be a database schema. The *base SPC (algebra) queries* and output arities are

*Input relation:* Expression $R$; with arity equal to $arity(R)$.

*Unary singleton constant:* Expression $\{\langle a\rangle\}$, where $a \in$ **dom**; with *arity* equal to 1.

The family of *SPC (algebra) queries* contains all base SPC queries and, for SPC queries $q_1, q_2$ with arities $\alpha_1, \alpha_2$, respectively,

*Selection:* $\sigma_{j=a}(q_1)$ and $\sigma_{j=k}(q_1)$ whenever $j, k \leq \alpha_1$ and $a \in$ **dom**; these have arity $\alpha_1$.

*Projection:* $\pi_{j_1,\ldots,j_n}(q_1)$, where $j_1, \ldots, j_n \leq \alpha_1$; this has arity $n$.

*Cross product:* $q_1 \times q_2$; this has arity $\alpha_1 + \alpha_2$.

In practice, we sometimes use brackets to surround algebraic queries, such as $[R \times \sigma_{1=a}(S)](\mathbf{I})$. In addition, parentheses may be dropped if no ambiguity results.

The *semantics* of these queries is defined in the natural manner (see Exercise 4.8).

The SPC algebra includes unsatisfiable queries, such as $\sigma_{1=a}(\sigma_{1=b}(R))$, where $arity(R) \geq 1$ and $a \neq b$. This is equivalent to $q^{\emptyset}$.

As explored in Exercise 4.22, permitting as base SPC queries constant queries that are not unary (i.e., expressions of the form $\{\langle a_1\rangle, \ldots, \langle a_n\rangle\}$) yields expressive power greater than the rule-based conjunctive queries with equality. This is also true of selection formulas in which disjunction is permitted. As will be seen in Section 4.5, these capabilities

are subsumed by including an explicit union operator into the SPC algebra. Permitting negation in selection formulas also extends the expressive power of the SPC algebra (see Exercise 4.27b).

Before leaving SPC algebra, we mention three operators that can be simulated by the primitive ones. The first is intersection ($\cap$), which is easily simulated (see Exercise 4.28). The other two operators involve generalizations of the selection and cross-product operators. The resulting algebra is called the *generalized SPC algebra*. We shall introduce a normal form for generalized SPC algebra expressions.

The first operator is a generalization of selection to permit the specification of multiple conditions. A *positive conjunctive selection formula* is a conjunction $F = \gamma_1 \wedge \cdots \wedge \gamma_n$ ($n \geq 1$), where each conjunct $\gamma_i$ has the form $j = a$ or $j = k$ for positive integers $j, k$ and $a \in \mathbf{dom}$; and a *positive conjunctive selection operator* is an expression of the form $\sigma_F$, where $F$ is a positive conjunctive selection formula. The intended typing and semantics for these operators is clear, as is the fact that they can be simulated by a composition of selections as defined earlier.

The second operator, called *equi-join*, is a binary operator that combines cross-product and selection. A (well-formed) equi-join operator is an expression of the form $\bowtie_F$ where $F = \gamma_1 \wedge \cdots \wedge \gamma_n$ ($n \geq 1$) is a conjunction such that each conjunct $\gamma_i$ has the form $j = k$. An equi-join operator $\bowtie_F$ can be applied to any pair $I$, $J$ of relation instances, where the $arity(I) \geq$ the maximum integer occurring on the left-hand side of any equality in $F$, and $arity(J) \geq$ the maximum integer occurring on the right-hand side of any equality in $F$. Given an equi-join expression $I \bowtie_F J$, let $F'$ be the result of replacing each condition $j = k$ in $F$ by $j = arity(I) + k$. Then the semantics of $I \bowtie_F J$ is given by $\sigma_{F'}(I \times J)$. As with cross-product, equi-join is also defined for pairs of tuples, with an undefined output if the tuples do not satisfy the conditions specified.

We now develop a normal form for SPC algebra. We stress that this normal form is useful for theoretical purposes and, in general, represents a costly way to compute the answer of a given query (see Chapter 6).

An SPC algebra expression is in *normal form* if it has the form

$$\pi_{j_1,\ldots,j_n}(\{\langle a_1\rangle\} \times \cdots \times \{\langle a_m\rangle\} \times \sigma_F(R_1 \times \cdots \times R_k)),$$

where $n \geq 0$; $m \geq 0$; $a_1, \ldots, a_m \in \mathbf{dom}$; $\{1, \ldots, m\} \subseteq \{j_1, \ldots, j_n\}$; $R_1, \ldots, R_k$ are relation names (repeats permitted); and $F$ is a positive conjunctive selection formula.

**Proposition 4.4.2** For each (generalized) SPC query $q$ there is a generalized SPC query $q'$ in normal form such that $q \equiv q'$.

The proof of this proposition (see Exercise 4.12) is based on repeated application of the following eight *equivalence-preserving SPC algebra rewrite rules* (or *transformations*).

*Merge-select:* replace $\sigma_F(\sigma_{F'}(q))$ by $\sigma_{F \wedge F'}(q)$.

*Merge-project:* replace $\pi_{\vec{j}}(\pi_{\vec{k}}(q))$ by $\pi_{\vec{l}}(q)$, where $l_i = k_{j_i}$ for each term $l_i$ in $\vec{l}$.

*Push-select-through-project:* replace $\sigma_F(\pi_{\vec{j}}(q))$ by $\pi_{\vec{j}}(\sigma_{F'}(q))$, where $F'$ is obtained from $F$ by replacing all coordinate values $i$ by $j_i$.

*Push-select-through-singleton:* replace $\sigma_{1=j}(\langle a \rangle \times q)$ by $\langle a \rangle \times \sigma_{(j-1)=a}(q)$.

*Associate-cross:* replace $((q_1 \times \cdots \times q_n) \times q)$ by $(q_1 \times \cdots q_n \times q)$, and replace $(q \times (q_1 \times \cdots q_n))$ by $(q \times q_1 \times \cdots q_n)$.

*Commute-cross:* replace $(q \times q')$ by $\pi_{\vec{j}\,\vec{j'}}(q' \times q)$, where $\vec{j} = arity(q') + 1, \ldots, arity(q') + arity(q)$, and $\vec{j'} = 1, \ldots, arity(q')$.

*Push-cross-through-select:* replace $(\sigma_F(q) \times q')$ by $\sigma_F(q \times q')$, and replace $(q \times \sigma_F(q'))$ by $\sigma_{F'}(q \times q')$, where $F'$ is obtained from $F$ by replacing all coordinate values $i$ by $i + arity(q)$.

*Push-cross-through-project:* replace $(\pi_{\vec{j}}(q) \times q')$ by $\pi_{\vec{j}}(q \times q')$, and replace $(q \times \pi_{\vec{j}}(q'))$ by $\pi_{\vec{j'}}(q \times q')$, where $\vec{j'}$ is obtained from $\vec{j}$ by replacing all coordinate values $i$ by $i + arity(q)$.

For a set $\mathcal{S}$ of rewrite rules and algebra expressions $q, q'$, write $q \to_{\mathcal{S}} q'$, or simply $q \to q'$ if $\mathcal{S}$ is understood from the context, if $q'$ is the result of replacing a subexpression of $q$ according to one of the rules in $\mathcal{S}$. Let $\overset{*}{\to}_{\mathcal{S}}$ denote the reflexive, transitive closure of $\to_{\mathcal{S}}$.

A family $\mathcal{S}$ of rewrite rules is *sound* if $q \to_{\mathcal{S}} q'$ implies $q \equiv q'$. If $\mathcal{S}$ is sound, then clearly $q \overset{*}{\to}_{\mathcal{S}} q'$ implies $q \equiv q'$.

It is easily verified that the foregoing set of rewrite rules is sound and that for each SPC query $q$ there is a normal form SPC query $q'$ such that $q'$ is in normal form, and $q \overset{*}{\to} q'$ (see Exercise 4.12).

In Section 6.1, we describe an approach to optimizing the evaluation of conjunctive queries using rewrite rules. For example, in that context, the merge-select and merge-project transformations are helpful, as are the *inverses* of the push-cross-through-select and push-cross-through-project.

Finally, note that an SPC query may require, as the result of transitivity, the equality of two distinct constants. Thus there are unsatisfiable SPC queries equivalent to $q^{\emptyset}$. This is analogous to the logic-based conjunctive queries with equality. It is clear, using the normal form, that one can check whether an SPC query is $q^{\emptyset}$ by examining the selection formula $F$. The set of SPC queries that are not equivalent to $q^{\emptyset}$ forms the *satisfiable SPC algebra*.

### The Named Perspective: The SPJR Algebra

In Example 4.4.1, the relation $I_3$ was constructed using selection and cross-product by the expression $\sigma_{1=5}(I_1 \times Pariscope)$. As is often the case, the columns used in this selection are labeled by the same attribute. In the context of the named perspective on tuples, this suggests a natural variant of the cross-product operator (and of the equi-join operator) that is called *natural join* and is denoted by $\bowtie$. Informally, the natural join requires the tuples that are concatenated to agree on the common attributes.

**EXAMPLE 4.4.3** The natural join of *Movies* and *Pariscope* is

$$\begin{aligned}
&\mathit{Movies} \bowtie \mathit{Pariscope} \\
&\quad = \{u \text{ with sort } \mathit{Title\ Director\ Actor\ Theater\ Schedule} \mid \\
&\qquad\qquad \text{for some } v \in \text{Movies and } w \in \text{Pariscope}, \\
&\qquad\qquad u[\mathit{Title\ Director\ Actor}] = v \text{ and } u[\mathit{Theater\ Title\ Schedule}] = w\} \\
&\quad = \pi_{1,2,3,4,6}(\mathit{Movies} \bowtie_{1=2} \mathit{Pariscope})
\end{aligned}$$

(assuming that the sort of the last expression corresponds to that of the previous expression). More generally, using the natural analog of projection and selection for the named perspective, query (4.4) can be expressed as

$$\pi_{\mathit{Theater},\mathit{Address}}((\sigma_{\mathit{Director}=\text{“Bergman”}}(\mathit{Movies}) \bowtie \mathit{Pariscope}) \bowtie \mathit{Location}).$$

As suggested by the preceding example, natural join can be used in the named context to replace certain equi-joins arising in the unnamed context. However, a problem arises if two relations sharing an attribute $A$ are to be joined but without forcing equality on the $A$ coordinates, or if a join is to be formed based on the equality of attributes not sharing the same name. For example, consider the query

**(4.8)** List pairs of actors that acted in the same movie.

To answer this, one would like to join the *Movies* relation with itself but matching only on the *Title* column. This will be achieved by first creating a copy *Movies′* of *Movies* in which the attribute *Director* has been renamed to *Director′* and *Actor* to *Actor′*; joining this with *Movies*; and finally projecting onto the *Actor* and *Actor′* columns. Renaming is also needed for query (4.6) (see Exercise 4.11).

The *named conjunctive algebra* has four primitive operators: *selection*, essentially as before; *projection*, now with repeats not permitted; (*natural*) *join*; and *renaming*. It is thus referred to as the *SPJR algebra*. As with the SPC algebra, we define the individual operators and then indicate how they are combined to form a typed, polymorphic algebra. In each case, we indicate the sorts of input and output. If a relation name is needed for the output, then it is assumed to be chosen to have the correct sort.

*Selection:* The selection operators have the form $\sigma_{A=a}$ and $\sigma_{A=B}$, where $A, B \in \mathbf{att}$ and $a \in \mathbf{dom}$. These operators apply to any instance $I$ with $A \in \mathit{sort}(I)$ [respectively, $A, B \in \mathit{sort}(I)$] and are defined in analogy to the unnamed selection, yielding an output with the same sort as the input.

*Projection:* The projection operator has the form $\pi_{A_1,\ldots,A_n}$, $n \geq 0$ (repeats not permitted) and operates on all inputs having sort containing $\{A_1, \ldots, A_n\}$, producing output with sort $\{A_1, \ldots, A_n\}$.

(*Natural*) *join:* This operator, denoted $\bowtie$, takes arbitrary inputs $I$ and $J$ having sorts $V$ and

$W$, respectively, and produces an output with sort equal to $V \cup W$. In particular,

$$I \bowtie J = \{t \text{ over } V \cup W \mid \text{for some } v \in I \text{ and } w \in J, \\ t[V] = v \text{ and } t[W] = w\}.$$

When $sort(I) = sort(J)$, then $I \bowtie J = I \cap J$, and when $sort(I) \cap sort(J) = \emptyset$, then $I \bowtie J$ is the cross-product of $I$ and $J$. The join operator is associative, commutative, and has the nonempty 0-ary relation $\{\langle\rangle\}$ as left and right identity. Because it is associative, we sometimes view join as a polyadic operator and write, for example, $I_1 \bowtie \cdots \bowtie I_n$.

As with cross-product and equi-join, natural join is extended to operate on pairs of tuples, with an undefined result if the tuples do not match on the appropriate attributes.

*Renaming:* An *attribute renaming* for a finite set $U$ of attributes is a one-one mapping from $U$ to **att**. An attribute renaming $f$ for $U$ can be described by specifying the set of pairs $(A, f(A))$, where $f(A) \neq A$; this is usually written as $A_1A_2 \ldots A_n \to B_1B_2 \ldots B_n$ to indicate that $f(A_i) = B_i$ for each $i \in [1, n]$ $(n \geq 0)$. A *renaming operator* for inputs over $U$ is an expression $\delta_f$, where $f$ is an attribute renaming for $U$; this maps to outputs over $f[U]$. In particular, for $I$ over $U$,

$$\delta_f(I) = \{v \text{ over } f[U] \mid \text{for some } u \in I,\ v(f(A)) = u(A) \text{ for each } A \in U\}.$$

---

**Example 4.4.4** Let $I$, $J$ be the two relations, respectively over $R$, $S$, given in Fig. 4.4. Then $I \bowtie J$, $\sigma_{A=1}(I)$, $\delta_{BC \to B'A}(J)$, and $\pi_A(I)$ are also shown there. Let $K$ be the one-tuple relation $\langle A : 1, C : 9\rangle$. Then $\pi_{A,B}(I \bowtie K)$ coincides with $\sigma_{A=1}(I)$ and $J \bowtie K = \{\langle A : 1, B : 8, C : 9\rangle\}$.

---

The *base* SPJR algebra queries are:

*Input relation:* Expression $R$; with *sort* equal to $sort(R)$.

*Unary singleton constant:* Expression $\{\langle A : a\rangle\}$, where $a \in$ **dom**; with *sort* $A$.

The remainder of the syntax and semantics of the SPJR algebra is now defined in analogy to those of the SPC algebra (see Exercise 4.8).

---

**Example 4.4.5** Consider again Fig. 4.4. Let $\mathbf{I}$ be the instance over $\{R, S\}$ such that $\mathbf{I}(R) = I$ and $\mathbf{I}(S) = J$. Then $[R]$ is a query and the answer to that query, denoted $R(\mathbf{I})$, is just $I$. Figure 4.4 also gives the values of $S(\mathbf{I})$, $[R \bowtie S](\mathbf{I})$, $[\sigma_{A=1}(R)](\mathbf{I})$, $[\delta_{BC \to B'A}(S)](\mathbf{I})$, and $[\pi_A(R)](\mathbf{I})$. Let $K_A = \{\langle A : 1\rangle\}$ and $K_C = \{\langle C : 9\rangle\}$. Then $[K_A]$ and $[K_C]$ are constant queries, and $[K_A \bowtie K_C]$ is a query that evaluates (on all inputs) to the relation $K$ of Example 4.4.4.

---

As with the SPC algebra, we introduce a natural generalization of the selection operator for the SPJR algebra. In particular, the notions of *positive conjunctive selection formula* and *positive conjunctive selection operator* are defined for the context in complete

$R$

| $A$ | $B$ |
|---|---|
| 1 | 2 |
| 4 | 2 |
| 6 | 6 |
| 7 | 7 |
| 1 | 7 |
| 1 | 6 |

$S$

| $B$ | $C$ |
|---|---|
| 2 | 3 |
| 2 | 5 |
| 6 | 4 |
| 8 | 9 |

$[R \bowtie S]$

| $A$ | $B$ | $C$ |
|---|---|---|
| 1 | 2 | 3 |
| 1 | 2 | 5 |
| 4 | 2 | 3 |
| 4 | 2 | 5 |
| 6 | 6 | 4 |
| 1 | 6 | 4 |

$[\sigma_{A=1}(R)]$

| $A$ | $B$ |
|---|---|
| 1 | 2 |
| 1 | 7 |
| 1 | 6 |

$[\delta_{BC \to B'A}(S)]$

| $B'$ | $A$ |
|---|---|
| 2 | 3 |
| 2 | 5 |
| 6 | 4 |
| 8 | 9 |

$[\pi_A(R)]$

| $A$ |
|---|
| 1 |
| 4 |
| 6 |
| 7 |

**Figure 4.4:** Examples of SPJR operators

analogy to the unnamed case. Including this operator yields the *generalized SPJR algebra*.

A normal form result analogous to that for the SPC algebra is now developed. In particular, an SPJR algebra expression is in *normal form* if it has the form

$$\pi_{B_1,\ldots,B_n}(\{\langle A_1 : a_1\rangle\} \bowtie \cdots \bowtie \{\langle A_m : a_m\rangle\} \bowtie \sigma_F(\delta_{f_1}(R_1) \bowtie \cdots \bowtie \delta_{f_k}(R_k))),$$

where $n \geq 0$; $m \geq 0$; $a_1, \ldots, a_m \in \mathbf{dom}$; each of $A_1, \ldots, A_m$ occurs in $B_1, \ldots, B_n$; the $A_i$'s are distinct; $R_1, \ldots, R_k$ are relation names (repeats permitted); $\delta_{f_j}$ is a renaming operator for $sort(R_j)$ for each $j \in [1, k]$ and no $A_i$'s occur in any $\delta_{f_j}(R_j)$; the sorts of $\delta_{f_1}(R_1), \ldots, \delta_{f_k}(R_k)$ are pairwise disjoint; and $F$ is a positive conjunctive selection formula. The following is easily verified (see Exercise 4.12).

**Proposition 4.4.6** For each (generalized) SPJR query $q$, there is a generalized SPJR query $q'$ in normal form such that $q \equiv q'$.

The set of SPJR queries not equivalent to $q^{\emptyset}$ forms the *satisfiable SPJR algebra*.

### Equivalence Theorem

We now turn to the main result of the chapter, showing the equivalence of the various formalisms introduced so far for expressing conjunctive queries. As shown earlier, the three logic-based versions of the conjunctive queries are equivalent. We now show that the SPC and SPJR algebras are also equivalent to each other and then obtain the equivalence of the algebraic languages and the three logic-based languages.

**LEMMA 4.4.7** The SPC and SPJR algebras are equivalent.

*Crux* We prove the inclusion SPC algebra $\sqsubseteq$ SPJR algebra; the converse is similar (see Exercise 4.14). Let $q$ be the following normal form SPC query:

$$\pi_{j_1,\ldots,j_n}(\{\langle a_1\rangle\} \times \cdots \times \{\langle a_m\rangle\} \times \sigma_F(R_1 \times \cdots \times R_k)).$$

We now describe an SPJR query $q'$ that is equivalent to $q$; $q'$ has the following form:

$$\pi_{A_{j_1},\ldots,A_{j_n}}(\{\langle A_1 : a_1\rangle\} \bowtie \cdots \bowtie \{\langle A_m : a_m\rangle\} \bowtie \sigma_G(\delta_{f_1}(R_1) \bowtie \cdots \bowtie \delta_{f_k}(R_k))).$$

We use the renaming functions so that the attributes of $\delta_{f_t}(R_t)$ are $A_s, \ldots, A_{s'}$, where $s, \ldots, s'$ are the coordinate positions of $R_t$ in the expression $R_1 \times \cdots \times R_k$ and modify $F$ into $G$ accordingly. In a little more detail, for each $r \in [1, k]$ let $\beta(t) = m + \Sigma_{s=0}^{t} arity(R_s)$, and let $A_{m+1}, \ldots, A_{\beta(k)}$ be new attributes. For each $t \in [1, k]$, choose $\delta_{f_t}$ so that it maps the $i^{\text{th}}$ attribute of $R_t$ to the attribute $A_{\beta(t-1)+i}$. To define $G$, first define the function $\gamma$ from coordinate positions to attribute names so that $\gamma(j) = A_{m+j}$, extend $\gamma$ to be the identity on constants, and extend it further in the natural manner to map unnamed selection formulas to named selection formulas. Finally, set $G = \gamma(F)$. It is now straightforward to verify that $q' \equiv q$. ■

It follows immediately from the preceding lemma that the satisfiable SPC algebra and the satisfiable SPJR algebra are equivalent.

The equivalence between the two algebraic languages and the three logic-based languages holds with a minor caveat involving the empty query $q^{\emptyset}$. As noted earlier, the SPC and SPJR algebras can express $q^{\emptyset}$, whereas the logic-based languages cannot, unless extended with equality. Hence the equivalence result is stated for the satisfiable SPC and SPJR algebras.

Theorem 4.3.3 (i.e., the closure of the rule-based conjunctive queries under composition) is used in the proof of this result. The closures of the SPC and SPJR algebras under composition are, of course, immediate.

**THEOREM 4.4.8 (Equivalence Theorem)** The rule-based conjunctive queries, tableau queries, conjunctive calculus queries, satisfiable SPC algebra, and satisfiable SPJR algebra are equivalent.

*Proof* The proof can be accomplished using the following steps:

(i) satisfiable SPC algebra $\sqsubseteq$ rule-based conjunctive queries; and

(ii) rule-based conjunctive queries $\sqsubseteq$ satisfiable SPC algebra.

We briefly consider how steps (i) and (ii) might be demonstrated; the details are left to the reader (Exercise 4.15). For (i), it is sufficient to show that each of the SPC algebra operations can be simulated by a rule. Indeed, then the inclusion follows from the fact that rule-based conjunctive queries are closed under composition by Theorem 4.3.3 and that

satisfiable rules with equality can be expressed as rules without equality. The simulation of algebra operations by rules is as follows:

1. $P \times Q$, where $P$ and $Q$ are not constant relations, corresponds to $ans(\vec{x}, \vec{y}) \leftarrow P(\vec{x}), Q(\vec{y})$, where $\vec{x}$ and $\vec{y}$ contain no repeating variables; in the case when $P$ ($Q$) are constant relations, $\vec{x}$ ($\vec{y}$) are the corresponding constant tuples.
2. $\sigma_F(R)$ corresponds to $ans(\vec{x}) \leftarrow R(\sigma_F(\vec{y}))$, where $\vec{y}$ consists of distinct variables, $\sigma_F(\vec{y})$ denotes the vector of variables and constants obtained by merging variables of $\vec{y}$ with other variables or with constants according to the (satisfiable) selection formula $F$, and $\vec{x}$ consists of the distinct variables in $\sigma_F(\vec{y})$.
3. $\pi_{j_1 \ldots j_n}(R)$ corresponds to $ans(x_{j_1} \ldots x_{j_n}) \leftarrow R(x_1 \ldots x_m)$, where $x_1, \ldots, x_m$ are distinct variables.

Next consider step (ii). Let $ans(\vec{x}) \leftarrow R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$ be a rule. There is an equivalent SPC algebra query in normal form that involves the cross-product of $R_1, \ldots, R_n$, a selection reflecting the constants and repeating variables occurring in $\vec{x}_1, \ldots, \vec{x}_n$, a further cross-product with constant relations corresponding to the constants in $\vec{x}$, and finally a projection extracting the coordinates corresponding to $\vec{x}$. ■

An alternative approach to showing step (i) of the preceding theorem is explored in Exercise 4.18.

## 4.5 Adding Union

As indicated by their name, conjunctive queries are focused on selecting data based on a conjunction of conditions. Indeed, each atom added to a rule potentially adds a further restriction to the tuples produced by the rule. In this section we consider a natural mechanism for adding a disjunctive capability to the conjunctive queries. Specifically, we add a *union* operator to the SPC and SPJR algebras, and we add natural analogs of it to the rule-based and tableau-based paradigms. Incorporating union into the conjunctive calculus raises some technical difficulties that are resolved in Chapter 5. This section also considers the evaluation of queries with union and introduces a more restricted mechanism for incorporating a disjunctive capability.

We begin with some examples.

---

**EXAMPLE 4.5.1** Consider the following query:

**(4.10)** Where can I see "Annie Hall" or "Manhattan"?

Although this cannot be expressed as a conjunctive query (see Exercise 4.22), it is easily expressed if union is added to the SPJR algebra:

$$\pi_{Theater}(\sigma_{Title=\text{"Annie Hall"}}(Pariscope) \cup \sigma_{Title=\text{"Manhattan"}}(Pariscope)).$$

An alternative formulation of this uses an extended selection operator that permits disjunctions in the selection condition:

$$\pi_{Theater}(\sigma_{Title=\text{“Annie Hall”}\vee Title=\text{“Manhattan”}}(Pariscope)).$$

As a final algebraic alternative, this can be expressed in the original SPJR algebra but permitting nonsingleton constant relations as base expressions:

$$\pi_{Theater}(Pariscope \bowtie \{\langle Title\text{: “Annie Hall”}\rangle, \langle Title\text{: “Manhattan”}\rangle\}).$$

The rule-based formalism can accommodate this query by permitting more than one rule with the same relation name in the head and taking the union of their outputs as the answer:

$$ans(x_t) \leftarrow Pariscope(x_t, \text{“Annie Hall”}, x_s)$$
$$ans(x_t) \leftarrow Pariscope(x_t, \text{“Manhattan”}, x_s).$$

Consider now the following query:

**(4.11)** What are the films with Allen as actor or director?

This query can be expressed using any of the preceding formalisms, except for the SPJR algebra extended with nonsingleton constant relations as base expressions (see Exercise 4.22).

---

Let $I_1$, $I_2$ be two relations with the same arity. As standard in mathematics, $I_1 \cup I_2$ is the relation having this arity and containing the union of the two sets of tuples. The definition of the *SPCU algebra* is obtained by extending the definition of the SPC algebra to include the *union* operator. The *SPJRU algebra* is obtained in the same fashion, except that union can only be applied to expressions having the same sort.

The SPCU and SPJRU algebras can be generalized by extending the selection operator (and join, in the case of SPC) as before. We can then define *normal forms* for both algebras, which are expressions consisting of one or more normal form SPC (SPJR) expressions combined using a polyadic union operator (see Exercise 4.23). As suggested by the previous example, disjunction can also be incorporated into selection formulas with no increase in expressive power (see Exercise 4.22).

Turning now to rule-based conjunctive queries, the simplest way to incorporate the capability of union is to consider sets of rules all having the same relation name in the head. These queries are evaluated by taking the union of the output of the individual rules.

This can be generalized without increasing the expressive power by incorporating something analogous to query composition. A *nonrecursive datalog program* (*nr-datalog program*) over schema **R** is a set of rules

$$S_1 \leftarrow body_1$$
$$S_2 \leftarrow body_2$$
$$\vdots$$
$$S_m \leftarrow body_m,$$

where no relation name in $\mathbf{R}$ occurs in a rule head; the same relation name may appear in more than one rule head; and there is some ordering $r_1, \ldots, r_m$ of the rules so that the relation name in the head of $r_i$ does not occur in the body of a rule $r_j$ whenever $j \leq i$.

The term 'nonrecursive' is used because recursion is not permitted. A simple example of a recursive rule is

$$ancestor(x, z) \leftarrow parent(x, y),\ ancestor(y, z).$$

A fixpoint operator is used to give the semantics for programs involving such rules. Recursion is the principal topic of Part D.

As in the case of rule-based conjunctive query programs, the query is evaluated on input $\mathbf{I}$ by evaluating each rule in (one of) the order(s) satisfying the foregoing property and forming unions whenever two rules have the same relation name in their heads. Equality atoms can be added to these queries, as they were for the rule-based conjunctive queries.

In general, a nonrecursive datalog program $P$ over $\mathbf{R}$ is viewed as having a database schema as target. Program $P$ can also be viewed as mapping from $\mathbf{R}$ to a single relation (see Exercise 4.24).

Turning to tableau queries, a *union of tableaux query* over schema $\mathbf{R}$ (or $R$) is an expression of the form $(\{\mathbf{T}_1, \ldots, \mathbf{T}_n\}, u)$, where $n \geq 1$ and $(\mathbf{T}_i, u)$ is a tableau query over $\mathbf{R}$ for each $i \in [1, n]$. The semantics of these queries is obtained by evaluating the queries $(\mathbf{T}_i, u)$ independently and then taking the union of their results. Equality is incorporated into these queries by permitting each of the queries $(\mathbf{T}_i, u)$ to have equality.

We can now state (see Exercise 4.25) the following:

**THEOREM 4.5.2** The following have equivalent expressive power:

1. the nonrecursive datalog programs (with single relation target),
2. the SPCU queries,
3. the SPJRU queries.

The union of tableau queries is weaker than the aforementioned languages with union. This is essentially because the definition of union of tableau queries does not allow separate summary rows for each tableau in the union. With just one summary row, the nonrecursive datalog query

$$ans(a) \leftarrow$$
$$ans(b) \leftarrow$$

cannot be expressed as a union of tableaux query.

As with conjunctive queries, it is easy to show that the conjunctive queries with union and equality are closed under composition.

### Union and the Conjunctive Calculus

At first glance, it would appear that the power of union can be added to the conjunctive calculus simply by permitting *disjunction* (denoted $\vee$) along with conjunction as a binary connective for formulas. This approach, however, can have serious consequences.

---

**EXAMPLE 4.5.3** Consider the following "query":

$$q = \{x, y, z \mid R(x, y) \vee R(y, z)\}.$$

Speaking intuitively, the "answer" of $q$ on nonempty instance $I$ will be (using a slight abuse of notation)

$$q(I) = (I \times \mathbf{dom}) \cup (\mathbf{dom} \times I).$$

This is an infinite set of tuples and thus not an instance according to the formal definition.

---

Informally, the query $q$ of the previous example is not "safe." This notion is one of the central topics that needs to be resolved when using the first-order predicate calculus as a relational query language, and it is studied in Chapter 5. We return there to the issue of adding union to the conjunctive calculus (see also Exercise 4.26).

## Bibliographic Notes

Codd's pioneering article [Cod70] on the relational model introduces the first relational query language, a named algebra. The predicate calculus was adapted to the relational model in [Cod72b], where it was shown to be essentially equivalent to the algebra. The conjunctive queries, in the calculus paradigm, were first introduced in [CM77]. Their equivalence with the SPC algebra is also shown there.

Typed tableau queries appeared as a two-dimensional representation of a subset of the conjunctive queries in [ASU79b] along with a proof that all typed restricted SPJ algebra expressions over one relation can be expressed using them. A precursor to the typed tableau queries is found in [ABU79], which uses a technique related to tableaux to analyze the join operator. [ASU79a, ASSU81, CV81] continued the investigation of typed tableau queries; [SY80] extends tableau queries to include union and a limited form of difference; and [Klu88] extends them to include inequalities and order-based comparators. Tableau queries have also played an important role in dependency theory; this will be discussed in Part C.

Many of the results in this chapter (including, for example, the equivalence of the SPC and SPJR algebras and closure of conjunctive queries under composition) are essentially part of the folklore.

## Exercises

**Exercise 4.1** Express queries (4.1–4.3) and (4.5–4.9) as (a) rule-based conjunctive queries, (b) conjunctive calculus queries, (c) tableau queries, (d) SPC expressions, and (e) SPJR expressions.

**Exercise 4.2** Let $\mathbf{R}$ be a database schema and $q$ a rule.

(a) Prove that $q(\mathbf{I})$ is finite for each instance $\mathbf{I}$ over $\mathbf{R}$.

(b) Show an upper bound, given instance $\mathbf{I}$ of $\mathbf{R}$ and output arity for conjunctive query $q$, for the number of tuples that can occur in $q(\mathbf{I})$. Show that this bound can be achieved.

**Exercise 4.3** Let $\mathbf{R}$ be a database schema and $\mathbf{I}$ an instance of $\mathbf{R}$.

(a) Suppose that $\varphi$ is a conjunctive calculus formula over $\mathbf{R}$ and $\nu$ is a valuation for $free(\varphi)$. Prove that $\mathbf{I} \models \varphi[\nu]$ implies that the image of $\nu$ is contained in $adom(\mathbf{I})$.

(b) Prove that if $q$ is a conjunctive calculus query over $\mathbf{R}$, then only a finite number of valuations need to be considered when evaluating $q(\mathbf{I})$. (Note: The presence of existential quantifiers may have an impact on the set of valuations that need to be considered.)

**Exercise 4.4**

(a) Let $\varphi$ and $\psi$ be equivalent conjunctive calculus formulas, and suppose that $\Psi'$ is the result of replacing an occurrence of $\varphi$ by $\psi$ in conjunctive calculus formula $\Psi$. Prove that $\Psi$ and $\Psi'$ are equivalent.

(b) Prove that the application of the rewrite rules *rename* and *merge-exists* to a conjunctive calculus formula yields an equivalent formula.

(c) Prove that these rules can be used to transform any conjunctive calculus formula into an equivalent formula in normal form.

**Exercise 4.5**

(a) Formally define the syntax and semantics of rule-based conjunctive queries with equality and conjunctive calculus queries with equality.

(b) As noted in the text, logic-based conjunctive queries with equality can generally yield infinite answers if not properly restricted. Give a definition for *range-restricted* rule-based and conjunctive calculus queries with equality that ensures that queries satisfying this condition always yield a finite answer.

(c) Prove for each rule-based conjunctive query with equality $q$ that either $q \equiv q^{\emptyset}$ or $q \equiv q'$ for some rule-based conjunctive query $q'$ without equality. Give a polynomial time algorithm that decides whether $q \equiv q^{\emptyset}$, and if not, constructs an equivalent rule-based conjunctive query $q'$.

(d) Prove that each rule-based conjunctive query with equality but no constants is equivalent to a rule-based conjunctive query without equality.

**Exercise 4.6** Extend the syntax of the conjunctive calculus to include equality. Give a syntactic condition that ensures that the answer to a query $q$ on $\mathbf{I}$ involves only constants from $adom(q, \mathbf{I})$ and such that the answer can be obtained by considering only valuations whose range is contained in $adom(q, \mathbf{I})$.

**Exercise 4.7** Give a proof of Theorem 4.3.3.

**Exercise 4.8**

(a) Give a formal definition for the semantics of the SPC algebra.

(b) Give a formal definition for the syntax and semantics of the SPJR algebra.

**Exercise 4.9** Consider the algebra consisting of all SPJR queries in which constants do not occur.

(a) Define a normal form for this algebra.

(b) Is this algebra closed under composition?

(c) Is this algebra equivalent to the rule-based conjunctive queries without constants or equality?

**Exercise 4.10** Under the named perspective, a selection operator is *constant based* if it has the form $\sigma_{A=a}$, where $A \in \mathbf{att}$ and $a \in \mathbf{dom}$. Prove or disprove: Each SPJR algebra expression is equivalent to an SPJR algebra expression all of whose selection operators are constant based.

**Exercise 4.11** Prove that queries (4.6 and 4.8) cannot be expressed using the SPJ algebra (i.e., that renaming is needed).

**Exercise 4.12**

(a) Prove that the set of SPC transformations presented after the statement of Proposition 4.4.2 is sound (i.e., preserves equivalence).

(b) Prove Proposition 4.4.2.

(c) Prove that each SPJR query is equivalent to one in normal form. In particular, exhibit a set of equivalence-preserving SPJR algebra transformations used to demonstrate this result.

**Exercise 4.13**

(a) Prove that the nonempty 0-ary relation is the left and right identity for cross product and for natural join.

(b) Prove that for a fixed relation schema $S$, there is an identity for union for relations over $S$. What if $S$ is not fixed?

(c) Let $S$ be a relational schema. For the binary operations $\alpha \in \{\bowtie, \cup\}$, does there exist a relation $I$ such that $I \alpha J = I$ for each relation $J$ over $S$?

**Exercise 4.14** Complete the proof of Lemma 4.4.7 by showing the inclusion SPJR algebra $\sqsubseteq$ SPC algebra.

**Exercise 4.15**

(a) Prove Proposition 4.2.9.

(b) Complete the proof of Theorem 4.4.8.

**Exercise 4.16** Consider the problem of defining restricted versions of the SPC and SPJR algebras that are equivalent to the rule-based conjunctive queries without equality. Find natural restricted versions, or explain why they do not exist.

**Exercise 4.17** Let $q$ be a tableau query and $q'$ the SPC query corresponding to it via the translation sketched in Theorem 4.4.8. If $q$ has $r$ rows and $q'$ has $j$ joins of database (nonconstant) relations, show that $j = r - 1$.

♠ **Exercise 4.18**

(a) Develop an inductive algorithm that translates a satisfiable SPC query $q$ into a tableau query by associating a tableau query to each subquery of $q$.

(b) Do the same for SPJR queries.

(c) Show that if $q$ is a satisfiable SPC (SPRJ) query with $n$ joins (not counting joins involving constant relations), then the tableau of the corresponding tableau query has $n + 1$ rows.

♠ **Exercise 4.19** [ASU79b] This exercise examines the connection between typed tableaux and a subset of the SPJ algebra. A *typed restricted* SPJ algebra expression over $R$ is an SPJR algebra expression that uses only $[R]$ as base expressions and only constant-based selection (i.e., having the form $\sigma_{A=a}$ for constant $a$), projection, and (natural) join as operators.

(a) Describe a natural algorithm that maps typed restricted SPJ queries $q$ over $R$ into equivalent typed tableau queries $q' = (T, u)$ over $R$, where $|T| =$ (the number of join operations in $q$) $+ 1$.

(b) Show that $q = (\{\langle x, y_1\rangle, \langle x_1, y_1\rangle, \langle x_1, y\rangle\}, \langle x, y\rangle)$ is not the image of any typed restricted SPJ query under the algorithm of part (a).

★ (c) [ASSU81] Prove that the tableau query $q$ of part (b) is not equivalent to any typed restricted SPJ algebra expression.

**Exercise 4.20** [ASU79b] A typed tableau query $q = (T, u)$ with $T$ over relation $R$ is *repeat restricted* if

1. If $A \in sort(u)$, then no variable in $\pi_A(T) - \{u(A)\}$ occurs more than once in $T$.
2. If $A \notin sort(u)$, then at most one variable in $\pi_A(T)$ occurs more than once in $T$.

Prove that if $q = (T, u)$ is a typed repeat-restricted tableau query over $R$, then there is a typed restricted SPJ query $q'$ such that the image of $q'$ under the algorithm of Exercise 4.19 part (a) is $q$.

**Exercise 4.21** Extend Proposition 4.2.2 to include disjunction (i.e., union).

**Exercise 4.22** The following query is used in this exercise:

**(4.15)** Produce a binary relation that includes all tuples $\langle t$, "excellent"$\rangle$ where $t$ is a movie directed by Allen, and all tuples $\langle t$, "superb"$\rangle$ where $t$ is a movie directed by Hitchcock.

(a) Show that none of queries (4.10–4.15) can be expressed using the SPC or SPJR algebras.

A *positive selection formula* for the SPC and SPJR algebras is a selection formula as before, except that disjunction can be used in addition to conjunction. Define the *S+PC algebra* to be the SPC algebra extended to permit arbitrary positive selection operators; and define the *S+PJR algebra* analogously.

(b) Determine which of queries (4.10–4.15) can be expressed using the S+PJR algebra.

Define the *SPC-1* algebra* to be the SPC algebra, except that nonsingleton unary constant relations can be used as base queries; and define the *SPC-n* algebra* to be the SPC algebra,

except that nonsingleton constant relations of arbitrary arity can be used as base queries. Define the *SPJR-1*$^*$ and *SPJR-n*$^*$ algebras analogously.

(c) Determine which of queries (4.10–4.15) can be expressed using the SPJR-1$^*$ and SPJR-$n^*$ algebras.

(d) Determine the relative expressive powers of the S+PC, SPC-1$^*$, SPC-$n^*$, and SPCU algebras.

**Exercise 4.23** Give precise definitions for normal forms for the SPCU and SPJRU algebras, and prove that all expressions from these algebras have an equivalent in normal form.

**Exercise 4.24** An nr-datalog program $P$ is in *normal form* if all relation names in rule heads are identical. Prove that each nonrecursive datalog query with single relation target has an equivalent in normal form.

**Exercise 4.25** Prove Theorem 4.5.2.

★ **Exercise 4.26** Recall the discussion in Section 4.5 about disjunction in the conjunctive calculus.

(a) Consider the query $q = \{x|\varphi(x)\}$, where

$$\varphi(x) \equiv R(x) \wedge \exists y, z(S(y, x) \vee S(x, z)).$$

Let $\mathbf{I}$ be an instance over $\{R, S\}$. Using the natural extension of the notion of *satisfies* to disjunction, show for each subformula of $\varphi$ with form $\exists\omega\psi$, and each valuation $\nu$ over $free(\exists\omega\psi)$ with range contained in $adom(\mathbf{I})$ that: there exists $c \in \mathbf{dom}$ such that $\mathbf{I} \models \psi[\nu \cup \{w/c\}]$ iff there exists $c \in adom(\mathbf{I})$ such that $\mathbf{I} \models \psi[\nu \cup \{w/c\}]$. Conclude that this query can be evaluated by considering only valuations whose range is contained in $adom(\mathbf{I})$.

(b) The *positive existential* (*relational*) *calculus* is the relational calculus query language in which query formulas are constructed using $\wedge$, $\vee$, $\exists$. Define a condition on positive existential calculus queries that guarantees that the answer involves only constants from $adom(q, \mathbf{I})$ and such that the answer can be obtained by considering only valuations whose range is contained in $adom(q, \mathbf{I})$. Extend the restriction for the case when equality is allowed in the calculus.

(c) Prove that the family of restricted positive existential calculus queries defined in the previous part has expressive power equivalent to the rule-based conjunctive queries with union and that this result still holds if equality is added to both families of queries.

**Exercise 4.27**

(a) Consider as an additional algebraic operation, the *difference*. The semantics of $q - q'$ is given by $[q - q'](\mathbf{I}) = q(\mathbf{I}) - q'(\mathbf{I})$. Show that the difference cannot be simulated in the SPCU or SPJRU algebras. (*Hint:* Use the monotonicity property of these algebras.)

(b) Negation can be added to (generalized) selection formulas in the natural way—that is, if $\gamma$ is a selection formula, then so is $(\neg\gamma)$. Give a precise definition for the syntax and semantics of selection with negation. Prove that the SPCU algebra cannot simulate selections of the form $\sigma_{\neg 1=2}(R)$ or $\sigma_{\neg 1=a}(R)$.

**Exercise 4.28** Show that intersection can be expressed in the SPC algebra.

★ **Exercise 4.29**

(a) Prove that there is no redundant operation in the set $\chi = \{\sigma, \pi, \times, \cup\}$ of unnamed algebra operators (i.e., for each operator $\alpha$ in the set, exhibit a schema and an algebraic query $q$ over that schema such that $q$ cannot be expressed with $\chi - \{\alpha\}$).

(b) Prove the analogous result for the set of named operators $\{\sigma, \pi, \bowtie, \delta, \cup\}$.

**Exercise 4.30** An *inequality atom* is an expression of the form $x \neq y$ or $x \neq a$, where $x$, $y$ are variables and $a$ is a constant. Assuming that the underlying domain has a total order, a *comparison atom* is an expression of the form $x\theta y$, $x\theta a$, or $a\theta x$, where $\theta$ ranges over $<$, $\leq$, $>$, and $\geq$.

(a) Show that the family of rule-based conjunctive queries with equality and inequality strictly dominates the family of rule-based conjunctive queries with equality.

(b) Assuming that the underlying domain has a total order, describe the relationships between the expressive powers of the family of rule-based conjunctive queries with equality; the family of rule-based conjunctive queries with equality and inequality; the family of rule-based conjunctive queries with equality and comparison atoms; and the family of rule-based conjunctive queries with equality, inequality, and comparison atoms.

(c) Develop analogous extensions and results for tableau queries, the conjunctive calculus, and SPC and SPJR algebras.

★ **Exercise 4.31** For some films, we may not want to store any actor name. Add to the domain a constant $\perp$ meaning unknown information. Propose an extension of the SPJR queries to handle unknown information (see Chapter 19).