5 Adding Negation: Algebra and Calculus

Alice: *Conjunctive queries are great. But what if I want to see a movie that doesn't feature Woody Allen?*
Vittorio: *We have to introduce negation.*
Sergio: *It is basically easy.*
Riccardo: *But the calculus is a little feisty.*

As indicated in the previous chapter, the conjunctive queries, even if extended by union, cannot express queries such as the following:

(5.1) What are the Hitchcock movies in which Hitchcock did not play?

(5.2) What movies are featured at the Gaumont Opera but not at the Gaumont les Halles?

(5.3) List those movies for which all actors of the movie have acted under Hitchcock's direction.

This chapter explores how negation can be added to all forms of the conjunctive queries (except for the tableau queries) to provide the power needed to express such queries. This yields languages in the various paradigms that have the same expressive power. They include relational algebra, relational calculus, and nonrecursive datalog with negation. The class of queries they express is often referred to as the *first-order queries* because relational calculus is essentially first-order predicate calculus without function symbols. These languages are of fundamental importance in database systems. They provide adequate power for many applications and at the same time can be implemented with reasonable efficiency. They constitute the basis for the standard commercial relational languages, such as SQL.

In the case of the algebras, negation is added using the set difference operator, yielding the language(s) generally referred to as *relational algebra* (Section 5.1). In the case of the rule-based paradigm, we consider negative literals in the bodies of rules, which are interpreted as the absence of the corresponding facts; this yields *nonrecursive datalog*$^\neg$ (Section 5.2).

Adding negation in the calculus paradigm raises some serious problems that require effort and care to resolve satisfactorily. In the development in this chapter, we proceed in two stages. First (Section 5.3) we introduce the calculus, illustrate the problematic issues of "safety" and domain independence, and develop some simple solutions for them. We also show the equivalence between the algebra and the calculus at this point. The material in this section provides a working knowledge of the calculus that is adequate for understanding the study of its extensions in Parts D and E. The second stage in our study of the calculus

(Section 5.4) focuses on the important problem of finding syntactic restrictions on the calculus that ensure domain independence.

The chapter concludes with brief digressions concerning how aggregate functions can be incorporated into the algebra and calculus (Section 5.5), and concerning the emerging area of constraint databases, which provide a natural mechanism for representing and manipulating infinite databases in a finite manner (Section 5.6).

From the theoretical perspective, the most important aspects of this chapter include the demonstration of the equivalence of the algebra and calculus (including a relatively direct transformation of calculus queries into equivalent algebra ones) and the application of the classical proof technique of structural induction used on both calculus formulas and algebra expressions.

5.1 The Relational Algebras

Incorporating the *difference* operator, denoted '$-$', into the algebras is straightforward. As with union and intersection, this can only be applied to expressions that have the same sort, in the named case, or arity, in the unnamed case.

Example 5.1.1 In the named algebra, query (5.1) is expressed by

$$\pi_{Title}\sigma_{Director=\text{"Hitchcock"}}(Movies) - \pi_{Title}\sigma_{Actor=\text{"Hitchcock"}}(Movies).$$

The *unnamed relational algebra* is obtained by adding the difference operator to the SPCU algebra. It is conventional also to permit the *intersection* operator, denoted '$\cap$' in this algebra, because it is simulated easily using cross-product, select, and project or using difference (see Exercise 5.4). Because union is present, nonsingleton constant relations may be used in this algebra. Finally, the selection operator can be extended to permit negation (see Exercise 5.4).

The *named relational algebra* is obtained in an analogous fashion, and similar generalizations can be developed.

As shown in Exercise 5.5, the family of unnamed algebra operators $\{\sigma, \pi, \times, \cup, -\}$ is nonredundant, and the same is true for the named algebra operators $\{\sigma, \pi, \bowtie, \delta, \cup, -\}$. It is easily verified that the algebras are not monotonic, nor are all algebra queries satisfiable (see Exercise 5.6). In addition, the following is easily verified (see Exercise 5.7):

Proposition 5.1.2 The unnamed and named relational algebras have equivalent expressive power.

The notion of *composition* of relational algebra queries can be defined in analogy to the composition of conjunctive queries described in the previous chapter. It is easily verified that the relational algebras, and hence the other equivalent languages presented in this chapter, are closed under composition.

5.2 Nonrecursive Datalog with Negation

To obtain a rule-based language with expressive power equivalent to the relational algebra, we extend nonrecursive datalog programs by permitting negative literals in rule bodies. This yields the *nonrecursive datalog with negation* also denoted *nonrecursive datalog*$^\neg$ and *nr-datalog*$^\neg$.

A *nonrecursive datalog*$^\neg$ (*nr-datalog*$^\neg$) rule is a rule of the form

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

where S is a relation name, u is a free tuple of appropriate arity, and each L_i is a *literal* [i.e., an expression of the form $R(v)$ or $\neg R(v)$, where R is a relation name and v is a free tuple of appropriate arity and where S does not occur in the body]. This rule is *range restricted* if each variable x occurring in the rule occurs in at least one literal of the form $R(v)$ in the rule body. Unless otherwise specified, all datalog$^\neg$ rules considered are assumed to be range restricted.

To give the *semantics* of the foregoing rule q, let **R** be a relation schema that includes all of the relation names occurring in the body of the rule q, and let **I** be an instance of **R**. Then the *image* of **I** under q is

$$\begin{aligned} q(\mathbf{I}) = \{\nu(u) \mid \nu \text{ is a valuation and for each } i \in [1, n],\\ \nu(u_i) \in \mathbf{I}(R_i), \text{ if } L_i = R_i(u_i), \text{ and}\\ \nu(u_i) \notin \mathbf{I}(R_i), \text{ if } L_i = \neg R_i(u_i)\}. \end{aligned}$$

In general, this image can be expressed as a difference $q_1 - q_2$, where q_1 is an SPC query and q_2 is an SPCU query (see Exercise 5.9).

Equality may be incorporated by permitting literals of the form $s = t$ and $s \neq t$ for terms s and t. The notion of *range restriction* in this context is defined as it was for rule-based conjunctive queries with equality. The semantics are defined in the natural manner.

To obtain the full expressive power of the relational algebras, we must consider sets of nr-datalog$^\neg$ rules; these are analogous to the nr-datalog programs introduced in the previous chapter. A *nonrecursive datalog*$^\neg$ *program* (with or without equality) over schema **R** is a sequence

$$\begin{gathered} S_1 \leftarrow body_1 \\ S_2 \leftarrow body_2 \\ \vdots \\ S_m \leftarrow body_m \end{gathered}$$

of nr-datalog$^\neg$ rules, where no relation name in **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 a rule r_i does not occur in the body of a rule r_j whenever $j \leq i$. The *semantics* of these programs are entirely analogous to the semantics of nr-datalog programs. An *nr-datalog$^\neg$ query* is a query defined by some nr-datalog$^\neg$ program with a specified target relation.

Example 5.2.1 Assume that each movie in *Movies* has one director. Query (5.1) is answered by

$$\begin{aligned} ans(x) \leftarrow\ & Movies(x, \text{“Hitchcock”}, z), \\ & \neg Movies(x, \text{“Hitchcock”}, \text{“Hitchcock”}). \end{aligned}$$

Query (5.3) is answered by

$$\begin{aligned} Hitch\text{-}actor(z) &\leftarrow Movies(x, \text{“Hitchcock”}, z) \\ not\text{-}ans(x) &\leftarrow Movies(x, y, z),\ \neg Hitch\text{-}actor(z) \\ ans(x) &\leftarrow Movies(x, y, z),\ \neg not\text{-}ans(x). \end{aligned}$$

Care must be taken when forming nr-datalog$^\neg$ programs. Consider, for example, the following program, which forms a kind of merging of the first two rules of the previous program. (Intuitively, the first rule is a combination of the first two rules of the preceding program, using variable renaming in the spirit of Example 4.3.1.)

$$\begin{aligned} bad\text{-}not\text{-}ans(x) &\leftarrow Movies(x, y, z),\ \neg Movies(x', \text{“Hitchcock”}, z), \\ &\qquad Movies(x', \text{“Hitchcock”}, z'), \\ ans(x) &\leftarrow Movies(x, y, z),\ \neg bad\text{-}not\text{-}ans(x) \end{aligned}$$

Rather than expressing query (5.3), it expresses the following:

(5.3′) (Assuming that all movies have only one director) list those movies for which all actors of the movie acted in *all* of Hitchcock's movies.

It is easily verified that each nr-datalog$^\neg$ program with equality can be simulated by an nr-datalog$^\neg$ program not using equality (see Exercise 5.10). Furthermore (see Exercise 5.11), the following holds:

Proposition 5.2.2 The relational algebras and the family of nr-datalog$^\neg$ programs that have single relation output have equivalent expressive power.

5.3 The Relational Calculus

Adding negation in the calculus paradigm yields an extremely flexible query language, which is essentially the predicate calculus of first-order logic (without function symbols). However, this flexibility brings with it a nontrivial cost: If used without restriction, the calculus can easily express queries whose “answers” are infinite. Much of the theoretical development in this and the following section is focused on different approaches to make the calculus "safe" (i.e., to prevent this and related problems). Although considerable effort is required, it is a relatively small price to pay for the flexibility obtained.

This section first extends the syntax of the conjunctive calculus to the full calculus. Then some intuitive examples are presented that illustrate how some calculus queries can violate the principle of "domain independence." A variety of approaches have been developed to resolve this problem based on the use of both semantic and syntactic restrictions.

This section focuses on semantic restrictions. The first step in understanding these is a somewhat technical definition based on "relativized interpretation" for the semantics of (arbitrary) calculus queries; the semantics are defined relative to different "underlying domains" (i.e., subsets of **dom**). This permits us to give a formal definition of domain independence and leads to a family of different semantics for a given query.

The section closes by presenting the equivalence of the calculus under two of the semantics with the algebra. This effectively closes the issue of expressive power of the calculus, at least from a semantic point of view. One of the semantics for the calculus presented here is the "active domain" semantics; this is particularly convenient in the development of theoretical results concerning the expressive power of a variety of languages presented in Parts D and E.

As noted in Chapter 4, the calculus presented in this chapter is sometimes called the *domain calculus* because the variables range over elements of the underlying domain of values. Exercise 5.23 presents the *tuple calculus*, whose variables range over tuples, and its equivalence with the domain calculus and the algebra. The tuple calculus and its variants are often used in practice. For example, the practical languages SQL and Quel can be viewed as using tuple variables.

Well-Formed Formulas, Revisited

We obtain the relational calculus from the conjunctive calculus with equality by adding negation ($\neg$), disjunction ($\vee$), and universal quantification ($\forall$). (Explicit equality is needed to obtain the full expressive power of the algebras; see Exercise 5.12.) As will be seen, both disjunction and universal quantification can be viewed as consequences of adding negation, because $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$ and $\forall x\varphi \equiv \neg\exists x\neg\varphi$.

The formal definition of the syntax of the relational calculus is a straightforward extension of that for the conjunctive calculus given in the previous chapter. We include the full definition here for the reader's convenience. A *term* is a constant or a variable. For a given input schema **R**, the *base formulas* include, as before, atoms over **R** and equality (inequality) atoms of the form $e = e'$ ($e \neq e'$) for terms e, e'. The (*well-formed*) *formulas* of the relational calculus over **R** include the base formulas and formulas of the form

(a) $(\varphi \wedge \psi)$, where φ and ψ are formulas over **R**;

(b) $(\varphi \vee \psi)$, where φ and ψ are formulas over **R**;

(c) $\neg\varphi$, where φ is a formula over **R**;

(d) $\exists x\varphi$, where x is a variable and φ a formula over **R**;

(e) $\forall x\varphi$, where x is a variable and φ a formula over **R**.

As with conjunctive calculus,

$$\exists x_1, x_2, \ldots, x_m \varphi \text{ abbreviates } \exists x_1 \exists x_2 \ldots \exists x_m \varphi, \text{ and}$$
$$\forall x_1, x_2, \ldots, x_m \varphi \text{ abbreviates} \forall x_1 \forall x_2 \ldots \forall x_m \varphi.$$

It is sometimes convenient to view the binary connectives $\wedge$ and $\vee$ as polyadic connectives. In some contexts, $e \neq e'$ is viewed as an abbreviation of $\neg(e = e')$.

It is often convenient to include two additional logical connectives, *implies* ($\rightarrow$) and *is equivalent to* ($\leftrightarrow$). We view these as syntactic abbreviations as follows:

$$\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$$
$$\varphi \leftrightarrow \psi \equiv (\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi).$$

The notions of *free* and *bound* occurrences of variables in a formula, and of $\mathit{free}(\varphi)$ for formula φ, are defined analogously to their definition for the conjunctive calculus. In addition, the notion of *relational calculus query* is defined, in analogy to the notion of conjunctive calculus query, to be an expression of the form

$$\{\langle e_1, \ldots, e_m \rangle : A_1, \ldots, A_m \mid \varphi\}, \text{ in the named perspective,}$$
$$\{e_1, \ldots, e_m \mid \varphi\}, \text{ in the unnamed perspective,}$$
$$\text{or if the sort is understood from the context,}$$

where $e_1, \ldots, e_m$ are terms, repeats permitted, and where the set of variables occurring in $e_1, \ldots, e_m$ is exactly $\mathit{free}(\varphi)$.

Example 5.3.1 Suppose that each movie has just one director. Query (5.1) can be expressed in the relational calculus as

$$\{x_t \mid \exists x_a \mathit{Movies}(x_t, \text{“Hitchcock”}, x_a) \wedge$$
$$\neg \mathit{Movies}(x_t, \text{“Hitchcock”}, \text{“Hitchcock”})\}.$$

Query (5.3) is expressed by

$$\{x_t \mid \exists x_d, x_a\ \mathit{Movies}(x_t, x_d, x_a) \wedge$$
$$\forall y_a\ (\exists y_d \mathit{Movies}(x_t, y_d, y_a)$$
$$\rightarrow \exists z_t\ \mathit{Movies}(z_t, \text{“Hitchock”}, y_a))\}.$$

The first conjunct ensures that the variable x_t ranges over titles in the current value of *Movies*, and the second conjunct enforces the condition on actors of the movie identified by x_t.

“Unsafe” Queries

Before presenting the alternative semantics for the relational calculus, we present an intuitive indication of the kinds of problems that arise if the conventional definitions from predicate calculus are adapted directly to the current context.

The fundamental problems of using the calculus are illustrated by the following expressions:

$$\begin{aligned} &(\textit{unsafe-1}) \quad \{x \mid \neg \textit{Movies}(\text{“Cries and Whispers”}, \text{“Bergman”}, x)\} \\ &(\textit{unsafe-2}) \quad \{x, y \mid \textit{Movies}(\text{“Cries and Whispers”}, \text{“Bergman”}, x) \\ &\qquad\qquad\qquad \vee\ \textit{Movies}(y, \text{“Bergman”}, \text{“Ullman”})\}. \end{aligned}$$

If the usual semantics of predicate calculus are adapted directly to this context, then the query (*unsafe-1*) produces all tuples $\langle a \rangle$ where $a \in$ **dom** and $\langle$“Cries and Whispers”, “Bergman”, $a\rangle$ is not in the input. Because all input instances are by definition finite, the query yields an infinite set on all input instances. The same is true of query (*unsafe-2*), even though it does not use explicit negation.

An intuitively appealing approach to resolving this problem is to view the different relation columns as typed and to insist that variables occurring in a given column range over only values of the appropriate type. For example, this would imply that the answer to query (*unsafe-1*) is restricted to the set of actors. This approach is not entirely satisfactory because query answers now depend on the domains of the types. For example, different answers are obtained if the type *Actor* includes all and only the current actors [i.e., persons occurring in $\pi_{Actor}(\textit{Movies})$] or includes all current *and potential* actors. This illustrates that query (*unsafe-1*) is not independent of the underlying domain within which the query is interpreted (i.e., it is not “domain independent”). The same is true of query (*unsafe-2*).

Even if the underlying domain is finite, users will typically not know the exact contents of the domains used for each variable. In this case it would be disturbing to have the result of a user query depend on information not directly under the user’s control. This is another argument for permitting only domain-independent queries.

A related but more subtle problem arises with regard to the interpretation of quantified variables. Consider the query

$$(\textit{unsafe-3}) \quad \{x \mid \forall y R(x, y)\}.$$

The answer to this query is necessarily finite because it is a subset of $\pi_1(R)$. However, the query is not domain independent. To see why, note that if y is assumed to range over all of **dom**, then the answer is always the empty relation. On the other hand, if the underlying domain of interpretation is finite, it is possible that the answer will be nonempty. (This occurs, for example, if the domain is $\{1, \ldots, 5\}$, and the input for R is $\{\langle 3, 1\rangle, \ldots \langle 3, 5\rangle\}$.) So again, this query depends on the underlying domain(s) being used (for the different variables) and is not under the user’s control.

There is a further difficulty of a more practical nature raised by query (*unsafe-3*). Specifically, if the intuitively appealing semantics of the predicate calculus are used, then the naive approach to evaluating quantifiers leads to the execution of potentially infinite procedures. Although the proper answer to such queries can be computed in a finite manner (see Theorem 5.6.1), this is technically intricate.

The following example indicates how easy it is to form an unsafe query mistakenly in practice.

Example 5.3.2 Recall the calculus query answering query (5.3) in Example 5.3.1. Suppose that the first conjunct of that query is omitted to obtain the following:

$$\{x_t \mid \forall y_a(\exists y_d Movies(x_t, y_d, y_a) \\ \rightarrow \exists z_t Movies(z_t, \text{“Hitchcock”}, y_a))\}.$$

This query returns all titles of movies that have the specified property and also all elements of **dom** not occurring in $\pi_{Title}(Movies)$. Even if x_t were restricted to range over the set of actual and potential movie titles, it would not be domain independent.

Relativized Interpretations

We now return to the formal development. As the first step, we present a definition that will permit us to talk about calculus queries in connection with different underlying domains.

Under the conventional semantics associated with predicate calculus, quantified variables range over all elements of the underlying domain, in our case, **dom**. For our purposes, however, we generalize this notion to permit explicit specification of the underlying domain to use (i.e., over which variables may range).

A *relativized instance* over schema **R** is a pair (**d**, **I**), where **I** is an instance over **R** and $adom(\mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$. A calculus formula φ is *interpretable* over (**d**,**I**) if $adom(\varphi) \subseteq \mathbf{d}$. In this case, if ν is a valuation over $free(\varphi)$ with range contained in **d**, then **I** *satisfies* φ for ν *relative* to **d**, denoted $\mathbf{I} \models_{\mathbf{d}} \varphi[\nu]$, if

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

(b) $\varphi = (s = s')$ is an equality atom and $\nu(s) = \nu(s')$;

(c) $\varphi = (\psi \wedge \xi)$ and[1] $\mathbf{I} \models_{\mathbf{d}} \psi[\nu|_{free(\psi)}]$ and $\mathbf{I} \models_{\mathbf{d}} \xi[\nu|_{free(\xi)}]$;

(d) $\varphi = (\psi \vee \xi)$ and $\mathbf{I} \models_{\mathbf{d}} \psi[\nu|_{free(\psi)}]$ or $\mathbf{I} \models_{\mathbf{d}} \xi[\nu|_{free(\xi)}]$;

(e) $\varphi = \neg\psi$ and $\mathbf{I} \not\models_{\mathbf{d}} \psi[\nu]$ (i.e., $\mathbf{I} \models_{\mathbf{d}} \psi[\nu]$ does not hold);

(f) $\varphi = \exists x\psi$ and for some $c \in \mathbf{d}$, $\mathbf{I} \models_{\mathbf{d}} \psi[\nu \cup \{x/c\}]$; or

(g) $\varphi = \forall x\psi$ and for each $c \in \mathbf{d}$, $\mathbf{I} \models_{\mathbf{d}} \psi[\nu \cup \{x/c\}]$.

The notion of “satisfies . . . relative to” just presented is equivalent to the usual notion of satisfaction found in first-order logic, where the set **d** plays the role of the universe of discourse in first-order logic. In practical database settings it is most natural to assume that the underlying universe is **dom**; for this reason we use specialized terminology here.

Recall that for a query q and input instance **I**, we denote $adom(q) \cup adom(\mathbf{I})$ by $adom(q, \mathbf{I})$, and the notation $adom(\varphi, \mathbf{I})$ for formula φ is defined analogously.

We can now define the relativized semantics for the calculus. Let **R** be a schema, $q = \{e_1, \ldots, e_n \mid \varphi\}$ a calculus query over **R**, and (**d**, **I**) a relativized instance over **R**. Then

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

the *image* of **I** under q *relative to* **d** is

$$q_{\mathbf{d}}(\mathbf{I}) = \{\nu(\langle e_1, \ldots, e_n\rangle) \mid \mathbf{I} \models_{\mathbf{d}} \varphi[\nu],$$
$$\nu \text{ is a valuation over } \mathit{free}(\varphi) \text{ with range} \subseteq \mathbf{d}\}.$$

Note that if **d** is infinite, then this image may be an infinite set of tuples.

As a minor generalization, for arbitrary $\mathbf{d} \subseteq \mathbf{dom}$, the *image* of q on **I** relative to **d** is defined by[2]

$$q_{\mathbf{d}}(\mathbf{I}) = q_{\mathbf{d} \cup adom(q,\mathbf{I})}(\mathbf{I}).$$

Example 5.3.3 Consider the query

$$q = \{x \mid R(x) \wedge \exists y(\neg R(y) \wedge \forall z(R(z) \vee z = y))\}$$

Then

$$\begin{aligned} q_{\mathbf{dom}}(I) &= \{\} \text{ for any instance } I \text{ over } R \\ q_{\{1,2,3,4\}}(J_1) &= \{\} \text{ for } J_1 = \{\langle 1\rangle, \langle 2\rangle\} \text{ over } R \\ q_{\{1,2,3,4\}}(J_2) &= J_2 \text{ for } J_2 = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle\} \text{ over } R \\ q_{\{1,2,3,4\}}(J_3) &= \{\} \text{ for } J_3 = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 4\rangle\} \text{ over } R \\ q_{\{1,2,3,4\}}(J_4) &= J_4 \text{ for } J_4 = \{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 5\rangle\} \text{ over } R. \end{aligned}$$

This illustrates that under an interpretation relative to a set **d**, a calculus query q on input **I** may be affected by $|\mathbf{d} - adom(q, \mathbf{I})|$.

It is important to note that the semantics of algebra and datalog$^\neg$ queries q evaluated on instance **I** are independent of whether **dom** or some subset **d** satisfying $adom(q, \mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$ is used as the underlying domain.

The Natural and Active Domain Semantics for Calculus Queries

The relativized semantics for calculus formulas immediately yields two important semantics for calculus queries. The first of these corresponds most closely to the conventional interpretation of predicate calculus and is thus perhaps the intuitively most natural semantics for the calculus.

Definition 5.3.4 For calculus query q and input instance **I**, the *natural* (or *unrestricted*) interpretation of q on **I**, denoted $q_{nat}(\mathbf{I})$, is $q_{\mathbf{dom}}(\mathbf{I})$ if this is finite and is undefined otherwise.

[2] Unlike the convention of first-order logic, interpretations over an empty underlying domain are permitted; this arises only with empty instances.

The second interpretation is based on restricting quantified variables to range over the active domain of the query and the input. Although this interpretation is unnatural from the practical perspective, it has the advantage that the output is always defined (i.e., finite). It is also a convenient semantics for certain theoretical developments.

Definition 5.3.5 For calculus query q and input instance $\mathbf{I}$, the *active domain* interpretation of q on $\mathbf{I}$, denoted $q_{adom}(\mathbf{I})$, is $q_{adom(q,\mathbf{I})}(\mathbf{I})$. The family of mappings obtained from calculus queries under the active domain interpretation is denoted CALC$_{adom}$.

Example 5.3.6 Recall query (*unsafe-2*). Under the natural interpretation on input the instance $\mathbf{I}$ shown in Chapter 3, this query yields the undefined result. On the other hand, under the active domain interpretation this yields as output (written informally) ({actors in "Cries and Whispers"} $\times$ *adom*($\mathbf{I}$)) $\cup$ (*adom*($\mathbf{I}$) $\times$ {movies by Bergman featuring Ullman}), which is finite and defined.

Domain Independence

As noted earlier, there are two difficulties with the natural interpretation of the calculus from a practical point of view: (1) it is easy to write queries with undefined output, and (2) even if the output is defined, the naive approach to computing it may involve consideration of quantifiers ranging over an infinite set. The active domain interpretation solves these problems but generally makes the answer dependent on information (the active domain) not readily available to users. One approach to resolving this situation is to restrict attention to the class of queries that yield the same output on all possible underlying domains.

Definition 5.3.7 A calculus query q is *domain independent* if for each input instance $\mathbf{I}$, and each pair $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$, $q_{\mathbf{d}}(\mathbf{I}) = q_{\mathbf{d}'}(\mathbf{I})$. If q is domain independent, then the *image* of q on input instance $\mathbf{I}$, denoted simply $q(\mathbf{I})$, is $q_{\mathbf{dom}}(\mathbf{I})$ [or equivalently, $q_{adom}(\mathbf{I})$]. The family of mappings obtained from domain-independent calculus queries is denoted CALC$_{di}$.

In particular, if q is domain independent, then the output according to the natural interpretation can be obtained by computing the active domain interpretation. Thus,

Lemma 5.3.8 CALC$_{di}$ $\sqsubseteq$ CALC$_{adom}$.

Example 5.3.9 The two calculus queries of Example 5.3.1 are domain independent, and the query of Example 5.3.2 is not (see Exercise 5.15).

Equivalence of Algebra and Calculus

We now demonstrate the equivalence of the various languages introduced so far in this chapter.

Theorem 5.3.10 (Equivalence Theorem) The domain-independent calculus, the calculus under active domain semantics, the relational algebras, and the family of nr-datalog$^\neg$ programs that have single-relation output have equivalent expressive power.

Proposition 5.2.2 shows that nr-datalog$^\neg$ and the algebras have equivalent expressive power. In addition, Lemma 5.3.8 shows that $\text{CALC}_{di} \sqsubseteq \text{CALC}_{adom}$. To complete the proof, we demonstrate that

(i) algebra $\sqsubseteq \text{CALC}_{di}$ (Lemma 5.3.11)

(ii) $\text{CALC}_{adom} \sqsubseteq$ algebra (Lemma 5.3.12).

Lemma 5.3.11 For each unnamed algebra query, there is an equivalent domain-independent calculus query.

Proof Let q be an unnamed algebra query with arity n. We construct a domain-independent query $q' = \{x_1, \ldots, x_n \mid \varphi_q\}$ that is equivalent to q. The formula φ_q is constructed using an induction on subexpressions of q. In particular, for subexpression E of q, we define φ_E according to the following cases:

(a) E is R for some $R \in \mathbf{R}$: φ_E is $R(x_1, \ldots, x_{arity(R)})$.

(b) E is $\{u_1, \ldots, u_m\}$, where each u_j is a tuple of arity α: φ_E is

$$(x_1 = u_1(1) \wedge \cdots \wedge x_\alpha = u_1(\alpha)) \vee \cdots \vee (x_1 = u_m(1) \wedge \cdots \wedge x_\alpha = u_m(\alpha)).$$

(c) E is $\sigma_F(E_1)$: φ_E is $\varphi_{E_1} \wedge \psi_F$, where ψ_F is the formula obtained from F by replacing each coordinate identifier i by variable x_i.

(d) E is $\pi_{i_1,\ldots,i_n}(E_1)$: φ_E is

$$\exists y_{i_1}, \ldots, y_{i_n}((x_1 = y_{i_1} \wedge \cdots \wedge x_n = y_{i_n}) \wedge \exists y_{j_1} \ldots \exists y_{j_l} \varphi_{E_1}(y_1, \ldots, y_{arity(E_1)})),$$

where $j_1, \ldots, j_l$ is a listing of $[1, arity(E_1)] - \{i_1, \ldots, i_n\}$.

(e) E is $E_1 \times E_2$: φ_E is $\varphi_{E_1} \wedge \varphi_{E_2}(x_{arity(E_1)+1}, \ldots, x_{arity(E_1)+arity(E_2)})$.

(f) E is $E_1 \cup E_2$: φ_E is $\varphi_{E_1} \vee \varphi_{E_2}$.

(g) E is $E_1 - E_2$: φ_E is $\varphi_{E_1} \wedge \neg\varphi_{E_2}$.

We leave verification of this construction and the properties of q' to the reader (see Exercise 5.13a). ■

Lemma 5.3.12 For each calculus query q, there is a query in the unnamed algebra that is equivalent to q under the active domain interpretation.

Crux Let $q = \{x_1, \ldots, x_n \mid \varphi\}$ be a calculus query over $\mathbf{R}$. It is straightforward to develop a unary algebra query E_{adom} such that for each input instance $\mathbf{I}$,

$$E_{adom}(\mathbf{I}) = \{\langle a \rangle \mid a \in adom(q, \mathbf{I})\}.$$

Next an inductive construction is performed. To each subformula $\psi(y_1, \ldots, y_m)$ of φ this associates an algebra expression E_ψ with the property that (abusing notation slightly)

$$\{y_1, \ldots, y_m \mid \psi\}_{adom(q,\mathbf{I})}(\mathbf{I}) = E_\psi(\mathbf{I}) \cap (adom(q, \mathbf{I}))^m.$$

[This may be different from using the active domain semantics on ψ, because we may have $adom(\psi, \mathbf{I}) \subset adom(q, \mathbf{I})$.] It is clear that E_φ is equivalent to q under the active domain semantics.

We now illustrate a few cases of the construction of expressions E_ψ and leave the rest for the reader (see Exercise 5.13b). Suppose that ψ is a subformula of φ. Then E_ψ is constructed in the following manner:

(a) $\psi(y_1, \ldots, y_m)$ is $R(t_1, \ldots, t_l)$, where each t_i is a constant or in $\vec{y}$: Then $E_\psi \equiv \pi_{\vec{k}}(\sigma_F(R))$, where $\vec{k}$ and F are chosen in accordance with $\vec{y}$ and $\vec{t}$.

(b) $\psi(y_1, y_2)$ is $y_1 \neq y_2$: E_ψ is $\sigma_{1\neq 2}(E_{adom} \times E_{adom})$.

(c) $\psi(y_1, y_2, y_3)$ is $\psi'(y_1, y_2) \vee \psi''(y_2, y_3)$: E_ψ is $(E_{\psi'} \times E_{adom}) \cup (E_{adom} \times E_{\psi''})$.

(d) $\psi(y_1, \ldots, y_m)$ is $\neg\psi'(y_1, \ldots, y_m)$: E_ψ is $(E_{adom} \times \cdots \times E_{adom}) - E_{\psi'}$. ■

5.4 Syntactic Restrictions for Domain Independence

As seen in the preceding section, to obtain the natural semantics for calculus queries, it is desirable to focus on domain independent queries. However, as will be seen in the following chapter (Section 6.3), it is undecidable whether a given calculus query is domain independent. This has led researchers to develop syntactic conditions that ensure domain independence, and many such conditions have been proposed.

Several criteria affect the development of these conditions, including their generality, their simplicity, and the ease with which queries satisfying the conditions can be translated into the relational algebra or other lower-level representations. We present one such condition here, called "safe range," that is relatively simple but that illustrates the flavor and theoretical properties of many of these conditions. It will serve as a vehicle to illustrate one approach to translating these restricted queries into the algebra. Other examples are explored in Exercises 5.25 and 5.26; translations of these into the algebra are considerably more involved.

This section begins with a brief digression concerning equivalence preserving rewrite rules for the calculus. Next the family CALC$_{sr}$ of safe-range queries is introduced. It is shown easily that the algebra $\sqsubseteq$ CALC$_{sr}$. A rather involved construction is then presented for transforming safe-range queries into the algebra. The section concludes by defining a variant of the calculus that is equivalent to the conjunctive queries with union.

1	$\varphi \wedge \psi$	$\leftrightarrow$	$\psi \wedge \varphi$
2	$\psi_1 \wedge \cdots \wedge \psi_n \wedge (\psi_{n+1} \wedge \psi_{n+2})$	$\leftrightarrow$	$\psi_1 \wedge \cdots \wedge \psi_n \wedge \psi_{n+1} \wedge \psi_{n+2}$
3	$\varphi \vee \psi$	$\leftrightarrow$	$\psi \vee \varphi$
4	$\psi_1 \vee \cdots \vee \psi_n \vee (\psi_{n+1} \vee \psi_{n+2})$	$\leftrightarrow$	$\psi_1 \vee \cdots \vee \psi_n \vee \psi_{n+1} \vee \psi_{n+2}$
5	$\neg(\varphi \wedge \psi)$	$\leftrightarrow$	$(\neg\varphi) \vee (\neg\psi)$
6	$\neg(\varphi \vee \psi)$	$\leftrightarrow$	$(\neg\varphi) \wedge (\neg\psi)$
7	$\neg(\neg\varphi)$	$\leftrightarrow$	φ
8	$\exists x\varphi$	$\leftrightarrow$	$\neg\forall x\neg\varphi$
9	$\forall x\varphi$	$\leftrightarrow$	$\neg\exists x\neg\varphi$
10	$\neg\exists x\varphi$	$\leftrightarrow$	$\forall x\neg\varphi$
11	$\neg\forall x\varphi$	$\leftrightarrow$	$\exists x\neg\varphi$
12	$\exists x\varphi \wedge \psi$	$\leftrightarrow$	$\exists x(\varphi \wedge \psi)$ (x not free in ψ)
13	$\forall x\varphi \wedge \psi$	$\leftrightarrow$	$\forall x(\varphi \wedge \psi)$ (x not free in ψ)
14	$\exists x\varphi \vee \psi$	$\leftrightarrow$	$\exists x(\varphi \vee \psi)$ (x not free in ψ)
15	$\forall x\varphi \vee \psi$	$\leftrightarrow$	$\forall x(\varphi \vee \psi)$ (x not free in ψ)
16	$\exists x\varphi$	$\leftrightarrow$	$\exists y\varphi^x_y$ (y not free in φ)
17	$\forall x\varphi$	$\leftrightarrow$	$\forall y\varphi^x_y$ (y not free in φ)

Figure 5.1: Equivalence-preserving rewrite rules for calculus formulas

Equivalence-Preserving Rewrite Rules

We now digress for a moment to present a family of rewrite rules for the calculus. These preserve equivalence regardless of the underlying domain used to evaluate calculus queries. Several of these rules will be used in the transformation of safe-range queries into the algebra.

Calculus formulas φ, ψ over schema **R** are *equivalent*, denoted $\varphi \equiv \psi$, if for each **I** over **R**, **d** $\subseteq$ **dom**, and valuation ν with range $\subseteq$ **d**

$$\mathbf{I} \models_{\mathbf{d} \cup adom(\varphi,\mathbf{I})} \varphi[\nu] \text{ if and only if } \mathbf{I} \models_{\mathbf{d} \cup adom(\psi,\mathbf{I})} \psi[\nu].$$

(It is verified easily that this generalizes the notion of equivalence for conjunctive calculus formulas.)

Figure 5.1 shows a number of equivalence-preserving rewrite rules for calculus formulas. It is straightforward to verify that if ψ transforms to ψ' by a rewrite rule and if φ' is the result of replacing an occurrence of subformula ψ of φ by formula ψ', then $\varphi' \equiv \varphi$ (see Exercise 5.14).

Note that, assuming $x \notin free(\psi)$ and $y \notin free(\varphi)$,

$$\exists x\varphi \wedge \forall y\psi \equiv \exists x\forall y(\varphi \wedge \psi) \equiv \forall y\exists x(\varphi \wedge \psi).$$

Example 5.4.1 Recall from Chapter 2 that a formula φ is in *prenex normal form* (PNF) if it has the form $\%_1 x_1 \ldots \%_n x_n \psi$, where each $\%_i$ is either $\forall$ or $\exists$, and no quantifiers occur in ψ. In this case, ψ is called the *matrix* of formula φ.

A formula ψ without quantifiers or connectives $\rightarrow$ or $\leftrightarrow$ is in *conjunctive normal form* (CNF) if it has the form $\xi_1 \wedge \cdots \wedge \xi_m$ $(m \geq 1)$, where each conjunct ξ_j has the form $L_1 \vee \cdots \vee L_k$ $(k \geq 1)$ and where each L_l is a literal (i.e., atom or negated atom). Similarly, a formula ψ without quantifiers or connectives $\rightarrow$ or $\leftrightarrow$ is in *disjunctive normal form* (DNF) if it has the form $\xi_1 \vee \cdots \vee \xi_m$, where each disjunct ξ_j has the form $L_1 \wedge \cdots \wedge L_k$ where each L_l is a literal (i.e., atom or negated atom).

It is easily verified (see Exercise 5.14) that the rewrite rules can be used to transform an arbitrary calculus formula into an equivalent formula that is in PNF with a CNF matrix, and into an equivalent formula that is in PNF with a DNF matrix.

Safe-Range Queries

The notion of safe range is presented now in three stages, involving (1) a normal form called SRNF, (2) a mechanism for determining how variables are "range restricted" by subformulas, and (3) specification of a required global property of the formula.

During this development, it is sometimes useful to speak of calculus formulas in terms of their parse trees. For example, we will say that the formula $(R(x) \wedge \exists y(S(y,z)) \wedge \neg T(x,z))$ has 'and' or $\wedge$ as a root (which has an atom, an $\exists$, and a $\neg$ as children).

The normalization of formulas puts them into a form more easily analyzed for safety without substantially changing their syntactic structure. The following equivalence-preserving rewrite rules are used to place a formula into *safe-range normal form* (SRNF):

Variable substitution: This is from Section 4.2. It is applied until no distinct pair of quantifiers binds the same variable and no variable occurs both free and bound.

Remove universal quantifiers: Replace subformula $\forall \vec{x} \psi$ by $\neg \exists \vec{x} \neg \psi$. (This and the next condition can be relaxed; see Example 5.4.5.)

Remove implications: Replace $\psi \rightarrow \xi$ by $\neg \psi \vee \xi$, and similarly for $\leftrightarrow$.

Push negations: Replace

(i) $\neg\neg\psi$ by ψ

(ii) $\neg(\psi_1 \vee \cdots \vee \psi_n)$ by $(\neg\psi_1 \wedge \cdots \wedge \neg\psi_n)$

(iii) $\neg(\psi_1 \wedge \cdots \wedge \psi_n)$ by $(\neg\psi_1 \vee \cdots \vee \neg\psi_n)$

so that the child of each negation is either an atom or an existentially quantified formula.

Flatten 'and's, 'or's, and existential quantifiers: This is done so that no child of an 'and' is an 'and,' and similarly for 'or' and existential quantifiers.

The SRNF formula resulting from applying these rules to φ is denoted $\mathtt{SRNF}(\varphi)$. A formula φ (query $\{\vec{e} \mid \varphi\}$) is in SRNF if $\mathtt{SRNF}(\varphi) = \varphi$.

Example 5.4.2 The first calculus query of Example 5.3.1 is in SRNF. The second calculus query is not in SRNF; the corresponding SRNF query is

$$\{x_t \mid \exists x_d, x_a Movies(x_t, x_d, x_a) \land \neg\exists y_a(\exists y_d Movies(x_t, y_d, y_a) \land \neg\exists z_t Movies(z_t, \text{“Hitchcock”}, y_a))\}.$$

Transforming the query of Example 5.3.2 into SRNF yields

$$\{x_t \mid \neg\exists y_a(\exists y_d Movies(x_t, y_d, y_a) \land \neg\exists z_t Movies(z_t, \text{“Hitchcock”}, y_a))\}.$$

We now present a syntactic condition on SRNF formulas that ensures that each variable is “range restricted,” in the sense that its possible values all lie within the active domain of the formula or the input. If a quantified variable is not range restricted, or if one of the free variables is not range restricted, then the associated query is rejected. To make the definition, we first define the set of *range-restricted variables* of an SRNF formula using the following procedure, which returns either the symbol $\perp$, indicating that some quantified variable is not range restricted, or the set of free variables that is range restricted.

ALGORITHM 5.4.3 (Range restriction (rr))

Input: a calculus formula φ in SRNF
Output: a subset of the free variables of φ or[3] $\perp$

```
begin
  case $\varphi$ of

          $R(e_1, \ldots, e_n)$ : $rr(\varphi)$ = the set of variables in $\{e_1, \ldots, e_n\}$;
      $x = a$ or $a = x$ : $rr(\varphi) = \{x\}$;
             $\varphi_1 \land \varphi_2$ : $rr(\varphi) = rr(\varphi_1) \cup rr(\varphi_2)$;
          $\varphi_1 \land x = y$ : $rr(\varphi) = \begin{cases} rr(\psi) & \text{if } \{x, y\} \cap rr(\psi) = \emptyset, \\ rr(\psi) \cup \{x, y\} & \text{otherwise;} \end{cases}$
             $\varphi_1 \lor \varphi_2$ : $rr(\varphi) = rr(\varphi_1) \cap rr(\varphi_2)$;
                $\neg\varphi_1$ : $rr(\varphi) = \emptyset$;
              $\exists\vec{x}\varphi_1$ : if $\vec{x} \subseteq rr(\varphi_1)$
                      then $rr(\varphi) = rr(\varphi_1) - \vec{x}$
                      else return $\perp$

  end case
end ■
```

---

[3] In the following, for each Z, $\perp \cup Z = \perp \cap Z = \perp - Z = Z - \perp = \perp$. In addition, we show the case of binary ‘and’s, etc., but we mean this to include polyadic ‘and’s, etc. Furthermore, we sometimes use ‘$\vec{x}$’ to denote the set of variables occurring in $\vec{x}$.

Intuitively, the occurrence of a variable x in a base relation or in an atom of the form $x = a$ restricts that variable. This restriction is propagated through $\wedge$, possibly lost in $\vee$, and always lost in $\neg$. In addition, each quantified variable must be restricted by the subformula it occurs in.

A calculus query $\{u \mid \varphi\}$ is *safe range* if $rr(\texttt{SRNF}(\varphi)) = \mathit{free}(\varphi)$. The family of safe-range queries is denoted by CALC_{sr}.

Example 5.4.4 Recall Examples 5.3.1 and 5.4.2. The first query of Example 5.3.1 is safe range. The first query of Example 5.4.2 is also safe range. However, the second query of Example 5.4.2 is not because the free variable x_t is not range restricted by the formula.

Before continuing, we explore a generalization of the notion of safe range to permit universal quantification.

Example 5.4.5 Suppose that formula φ has a subformula of the form

$$\psi \equiv \forall \vec{x}(\psi_1(\vec{x}) \rightarrow \psi_2(\vec{y})),$$

where $\vec{x}$ and $\vec{y}$ might overlap. Transforming into SRNF (and assuming that the parent of ψ is not $\neg$), we obtain

$$\psi' \equiv \neg\exists \vec{x}(\psi_1(\vec{x}) \wedge \neg\psi_2(\vec{y})).$$

Now $rr(\psi')$ is defined iff

(a) $rr(\psi_1) = \vec{x}$, and

(b) $rr(\psi_2)$ is defined.

In this case, $rr(\psi') = \emptyset$. This is illustrated by the second query of Example 5.3.1, that was transformed into SRNF in Example 5.4.2.

Thus SRNF can be extended to permit subformulas that have the form of ψ without materially affecting the development.

The calculus query constructed in the proof of Lemma 5.3.11 is in fact safe range. It thus follows that the algebra $\sqsubseteq \text{CALC}_{sr}$.

As shown in the following each safe range query is domain independent (Theorem 5.4.6). For this reason, if q is safe range we generally use the natural interpretation to evaluate it; we may also use the active domain interpretation.

The development here implies that all of CALC_{sr}, CALC_{di}, and CALC_{adom} are equivalent. When the particular choice is irrelevant to the discussion, we use the term *relational calculus* to refer to any of these three equivalent query languages.

From Safe Range to the Algebra

We now present the main result of this section (namely, the translation of safe-range queries into the named algebra). Speaking loosely, this translation is relatively direct in the sense that the algebra query E constructed for calculus query q largely follows the structure of q. As a result, evaluation of E will in most cases be more efficient than using the algebra query that is constructed for q by the proof of Lemma 5.3.12.

Examples of the construction used are presented after the formal argument.

Theorem 5.4.6 CALC$_{sr}$ $\equiv$ the relational algebra. Furthermore, each safe-range query is domain independent.

The proof of this theorem involves several steps. As seen earlier, the algebra $\sqsubseteq$ CALC$_{sr}$. To prove the other direction, we develop a translation from safe-range queries into the named algebra. Because the algebra is domain independent, this will also imply the second sentence of the theorem.

To begin, let φ be a safe-range formula in SRNF. An occurrence of a subformula ψ in φ is *self-contained* if its root is $\wedge$ or if

(i) $\psi = \psi_1 \vee \cdots \vee \psi_n$ and $rr(\psi) = rr(\psi_1) = \cdots = rr(\psi_n) = \mathit{free}(\psi)$;

(ii) $\psi = \exists \vec{x} \psi_1$ and $rr(\psi) = \mathit{free}(\psi_1)$; or

(iii) $\psi = \neg\psi_1$ and $rr(\psi) = \mathit{free}(\psi_1)$.

A safe-range, SRNF formula φ is in[4] *relational algebra normal form* (RANF) if each subformula of φ is self-contained.

Intuitively, if ψ is a self-contained subformula of φ that does not have $\wedge$ as a root, then all free variables in ψ are range restricted within ψ. As we shall see, if φ is in RANF, this permits construction of an equivalent relational algebra query E_φ using an induction from leaf to root.

We now develop an algorithm RANF-ALG that transforms safe-range SRNF formulas into RANF. It is based on the following rewrite rules:

(R1) Push-into-or: Consider the subformula

$$\psi = \psi_1 \wedge \cdots \wedge \psi_n \wedge \xi,$$

where

$$\xi = \xi_1 \vee \cdots \vee \xi_m.$$

Suppose that $rr(\psi) = \mathit{free}(\psi)$, but $rr(\xi_1 \vee \cdots \vee \xi_m) \neq \mathit{free}(\xi_1 \vee \cdots \vee \xi_m)$. Nondeterministically choose a subset $i_1, \ldots, i_k$ of $1, \ldots, n$ such that

$$\xi' = (\xi_1 \wedge \psi_{i_1} \wedge \cdots \wedge \psi_{i_k}) \vee \cdots \vee (\xi_m \wedge \psi_{i_1} \wedge \cdots \wedge \psi_{i_k})$$

[4] This is a variation of the notion of RANF used elsewhere in the literature; see Bibliographic Notes.

satisfies $rr(\xi') = free(\xi')$. (One choice of $i_1, \ldots, i_k$ is to use all of $1, \ldots, n$; this necessarily yields a formula ξ' with this property.) Letting $\{j_1, \ldots, j_l\} = \{1, \ldots, n\} - \{i_1, \ldots, i_k\}$, set

$$\psi' = \texttt{SRNF}(\psi_{j_1} \wedge \cdots \wedge \psi_{j_l} \wedge \xi').$$

The application of SRNF to ξ' only has the effect of possibly renaming quantified variables[5] and of flattening the roots of subformulas $\xi_p \wedge \psi_{i_1} \wedge \cdots \wedge \psi_{i_k}$, where ξ_p has root $\wedge$; analogous remarks apply. The rewrite rule is to replace subformula ψ by ψ' and possibly apply SRNF to flatten an $\vee$, if both $l = 0$ and the parent of ψ is $\vee$.

(R2) Push-into-quantifier: Suppose that

$$\psi = \psi_1 \wedge \cdots \wedge \psi_n \wedge \exists \vec{x} \xi,$$

where $rr(\psi) = free(\psi)$, but $rr(\xi) \neq free(\xi)$. Then replace ψ by

$$\psi' = \texttt{SRNF}(\psi_{j_1} \wedge \cdots \wedge \psi_{j_l} \wedge \exists \vec{x} \xi'),$$

where

$$\xi' = \psi_{i_1} \wedge \cdots \wedge \psi_{i_k} \wedge \xi$$

and where $rr(\xi') = free(\xi')$ and $\{j_1, \ldots, j_l\} = \{1, \ldots, n\} - \{i_1, \ldots, i_k\}$. The rewrite rule is to replace ψ by ψ' and possibly apply SRNF to flatten an $\exists$.

(R3) Push-into-negated-quantifier: Suppose that

$$\psi = \psi_1 \wedge \cdots \wedge \psi_n \wedge \neg \exists \vec{x} \xi,$$

where $rr(\psi) = free(\psi)$, but $rr(\xi) \neq free(\xi)$. Then replace ψ by

$$\psi' = \texttt{SRNF}(\psi_1 \wedge \cdots \wedge \psi_n \wedge \neg \exists \vec{x} \xi'),$$

where

$$\xi' = \psi_{i_1} \wedge \cdots \wedge \psi_{i_k} \wedge \xi$$

and where $rr(\xi') = free(\xi')$ and $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$. That ψ' is equivalent to ψ follows from the observation that the propositional formulas $p \wedge q \wedge \neg r$ and $p \wedge q \wedge \neg(p \wedge r)$ are equivalent. The rewrite rule is to replace ψ by ψ'.

The algorithm RANF-ALG for applying these rewrite rules is essentially top-down and recursive. We sketch the algorithm now (see Exercise 5.19).

[5] It is assumed that under SRNF renamed variables are chosen so that they do not occur in the full formula under consideration.

Algorithm 5.4.7 (Relational Algebra Normal Form (`RANF-ALG`))

Input: a safe-range calculus formula φ in SRNF

Output: a RANF formula $\varphi' = \texttt{RANF}(\varphi)$ equivalent to φ

```
begin
  while some subformula $\psi$ (with its conjuncts possibly reordered) of $\varphi$ satisfies the
  premise of R1, R2, or R3
  do
    case R1: (left as exercise)
         R2: (left as exercise)
         R3: Let $\psi = \psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\exists\vec{x}\xi$
             and $\psi_{i_1}, \ldots, \psi_{i_k}$ satisfy the conditions of R3;
             $\alpha := \texttt{RANF}(\psi_1 \wedge \cdots \wedge \psi_n)$;
             $\beta := \texttt{RANF}(\texttt{SRNF}(\psi_{i_1} \wedge \cdots \wedge \psi_{i_k} \wedge \xi))$;
             $\psi' := \alpha \wedge \neg\exists\vec{x}\beta$;
             $\varphi :=$ result of replacing $\psi$ by $\psi'$ in $\varphi$;
    end case
  end while
end
```

The proof that these rewrite rules can be used to transform a safe-range SRNF formula into a RANF formula has two steps (see Exercise 5.19). First, a case analysis can be used to show that if safe-range φ in SRNF is not in RANF, then one of the rewrite rules (R1, R2, R3) can be applied. Second, it is shown that Algorithm 5.4.7 terminates. This is accomplished by showing that (1) each successfully completed call to RANF-ALG reduces the number of non-self-contained subformulas, and (2) if a call to RANF-ALG on ψ invokes other calls to RANF-ALG, the input to these recursive calls has fewer non-self-contained subformulas than does ψ.

We now turn to the transformation of RANF formulas into equivalent relational algebra queries. We abuse notation somewhat and assume that each variable is also an attribute. (Alternatively, a one-one mapping *var-to-att* : **var** $\to$ **att** could be used.) In general, given a RANF formula φ with free variables $x_1, \ldots, x_n$, we shall construct a named algebra expression E_φ over attributes $x_1, \ldots, x_n$ such that for each input instance $\mathbf{I}$, $E_\varphi(\mathbf{I}) = \{x_1, \ldots, x_n \mid \varphi\}(\mathbf{I})$. (The special case of queries $\{e_1, \ldots, e_n \mid \varphi\}$, where some of the e_i are constants, is handled by performing a join with the constants at the end of the construction.)

A formula φ is in *modified relational algebra normal form (modified RANF)* if it is RANF, except that each polyadic 'and' is ordered and transformed into binary 'and's, so that atoms $x = y$ ($x \neq y$) are after conjuncts that restrict one (both) of the variables involved and so that each free variable in a conjunct of the form $\neg\xi$ occurs in some preceding conjunct. It is straightforward to verify that each RANF formula can be placed into modified RANF. Note that each subformula of a modified RANF formula is self-contained.

Let RANF formula φ be fixed. The construction of E_φ is inductive, from leaf to root, and is sketched in the following algorithm. The special operator **diff**, on inputs R and S where $att(S) \subset att(R)$, is defined by

$$R \textbf{ diff } S = R - (R \bowtie S).$$

(Many details of this transformation, such as the construction of renaming function f, projection list $\vec{k}$, and selection formula F in the first entry of the case statement, are left to the reader; see Example 5.4.9 and Exercise 5.19.)

Algorithm 5.4.8 (Translation into the Algebra)

Input: a formula φ in modified RANF

Output: an algebra query E_φ equivalent to φ

```
begin
    case $\varphi$ of
$R(\vec{e})$    $\delta_f(\pi_{\vec{k}}(\sigma_F(R)))$

$x = a$    $\{\langle x : a\rangle\}$

$\psi \wedge \xi$    if $\xi$ is $x = x$, then $E_\psi$
         if $\xi$ is $x = y$ (with $x, y$ distinct), then
             $\sigma_{x=y}(E_\psi)$, if $\{x, y\} \subseteq free(\psi)$
             $\sigma_{x=y}(E_\psi \bowtie \delta_{x \to y} E_\psi)$, if $x \in free(\psi)$ and $y \notin free(\psi)$
             $\sigma_{x=y}(E_\psi \bowtie \delta_{y \to x} E_\psi)$, if $y \in free(\psi)$ and $x \notin free(\psi)$
         if $\xi$ is $x \neq y$, then $\sigma_{x \neq y}(E_\psi)$
         if $\xi = \neg\xi'$, then
             $E_\psi$ diff $E_{\xi'}$, if $free(\xi') \subset free(\psi)$
             $E_\psi - E_{\xi'}$, if $free(\xi') = free(\psi)$
         otherwise, $E_\psi \bowtie E_\xi$

$\neg\psi$    $\{\langle\rangle\} - E_\psi$
             (in the case that $\neg\psi$ does not have 'and' as parent)

$\psi_1 \vee \cdots \vee \psi_n$    $E_{\psi_1} \cup \cdots \cup E_{\psi_n}$

$\exists x_1, \ldots, x_n \psi(x_1, \ldots, x_n, y_1, \ldots, y_m)$
                 $\pi_{y_1,\ldots,y_m}(E_\psi)$
  end case
end
```

Finally, let $q = \{x_1, \ldots, x_n \mid \varphi\}$ be safe range. Because the transformations used for SRNF and RANF are equivalence preserving, without loss of generality we can assume that φ is in modified RANF. To conclude the proof of Theorem 5.4.6, it must be shown that q and E_φ are equivalent. In fact, it can be shown that for each instance $\mathbf{I}$ and each $\mathbf{d}$ satisfying $adom(q, \mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$,

$$q_{\mathbf{d}}(\mathbf{I}) = E_\varphi(\mathbf{I}).$$

This will also yield that q is domain independent.

Let $\mathbf{I}$ and $\mathbf{d}$ be fixed. A straightforward induction can be used to show that for each subformula $\psi(y_1, \ldots, y_m)$ of φ and each variable assignment ν with range $\mathbf{d}$,

$$\mathbf{I} \models_{\mathbf{d}} \psi[\nu] \Leftrightarrow \langle \nu(y_1), \ldots, \nu(y_m)\rangle \in E_\psi(\mathbf{I})$$

(see Exercise 5.19.) This completes the proof of Theorem 5.4.6.

Example 5.4.9 (a) Consider the query

$$q_1 = \{\langle a, x, y\rangle : A_1A_2A_3 \mid \exists z(P(x, y, z) \vee [R(x, y)\wedge \\ ([S(z) \wedge \neg T(x, z)] \vee [T(y, z)])])\}.$$

The formula of q_1 is in SRNF. Transformation into RANF yields

$$\exists z(P(x, y, z) \vee [R(x, y) \wedge S(z) \wedge \neg T(x, z)] \vee [R(x, y) \wedge T(y, z)]).$$

Assuming the schemas $P[B_1B_2B_3]$, $R[C_1C_2]$, $S[D]$, and $T[F_1F_2]$, transformation of this into the algebra yields

$$\begin{aligned} E = \pi_{x,y}(&\delta_{B_1B_2B_3\to xyz}(P) \\ &\cup ((\delta_{C_1C_2\to xy}(R) \bowtie \delta_{D\to z}(S)) \textbf{ diff } \delta_{F_1F_2\to yz}(T)) \\ &\cup (\delta_{C_1C_2\to xy}(R) \bowtie \delta_{F_1F_2\to yz}(T))).\end{aligned}$$

Finally, an algebra query equivalent to q_1 is

$$\{\langle A_1 : a\rangle\} \bowtie \delta_{xy\to A_2A_3}(E).$$

(b) Consider the query

$$q_2 = \{x \mid \exists y[R(x, y) \wedge \forall z(S(z, a) \to T(y, z)) \\ \wedge \exists v, w(\neg T(v, w) \wedge w = b \wedge v = x)]\}.$$

Transforming to SRNF, we have

$$\exists y[R(x, y) \wedge \neg\exists z(S(z, a) \wedge \neg T(y, z)) \wedge \exists v, w(\neg T(v, w) \wedge w = b \wedge v = x)].$$

Transforming to RANF and reordering the conjunctions, we obtain

$$\exists y[\exists v, w(R(x, y)\wedge w = b\wedge v = x\wedge\neg T(v, w))\wedge\neg\exists z(R(x, y)\wedge S(z, a)\wedge\neg T(y, z))].$$

Assuming schemas $R[A_1, A_2]$, $S[B_1, B_2]$, and $T[C_1, C_2]$, the equivalent algebra query is obtained using the program

$$E_1 := (\delta_{A_1A_2 \to xy}(R) \bowtie \{\langle w : b\rangle\});$$
$$E_2 := (\sigma_{v=x}(E_1 \bowtie \delta_{x \to v}(E_1))) \textbf{ diff } \delta_{C_1C_2 \to vw}(T);$$
$$E_3 := \pi_{x,y}(E_2);$$
$$E_4 := \pi_{x,y}(\delta_{A_1A_2 \to xy}(R) \bowtie \delta_{B_1 \to z}(\pi_{B_1}(\sigma_{B_2=a}(S))) \textbf{ diff } \delta_{C_1C_2 \to yz}(T));$$
$$E_5 := \pi_x(E_3 - E_4).$$

The Positive Existential Calculus

In Chapter 4, disjunction was incorporated into the rule-based conjunctive queries, and union was incorporated into the tableau, SPC, and SPJR queries. Incorporating disjunction into the conjunctive calculus was more troublesome because of the possibility of infinite "answers." We now apply the tools developed earlier in this chapter to remedy this situation.

A *positive existential* (*calculus*) *query* is a domain-independent calculus query $q = \{e_1, \ldots, e_n \mid \varphi\}$, possibly with equality, in which the only logical connectives are $\wedge$, $\vee$, and $\exists$. It is decidable whether a query q with these logical connectives is domain independent; and if so, q is equivalent to a safe-range query using only these connectives (see Exercise 5.16). The following is easily verified.

Theorem 5.4.10 The positive existential calculus is equivalent to the family of conjunctive queries with union.

5.5 Aggregate Functions

In practical query languages, the underlying domain is many-sorted, with sorts such as boolean, string, integer, or real. These languages allow the use of comparators such as $\leq$ between database entries in an ordered sort and "aggregate" functions such as **sum**, **count**, or **average** on numeric sorts. In this section, aggregate operators are briefly considered. In the next section, a novel approach for incorporating arithmetic constraints into the relational model will be addressed.

Aggregate operators operate on collections of domain elements. The next example illustrates how these are used.

Example 5.5.1 Consider a relation *Sales*[*Theater*, *Title*, *Date*, *Attendance*], where a tuple $\langle th, ti, d, a\rangle$ indicates that on date d a total of a people attended showings of movie ti at theater th. We assume that $\{Theater, Title, Date\}$ is a key, i.e., that two distinct tuples cannot share the same values on these three attributes. Two queries involving aggregate functions are

(5.4) For each theater, list the total number of movies that have been shown there.

(5.5) For each theater and movie, list the total attendance.

Informally, the first query might be expressed in a pidgin language as

$$\{\langle th, c\rangle \mid th \text{ is a theater occurring in } \mathit{Sales} \text{ and } c = |\pi_{\mathit{Title}}(\sigma_{\mathit{Theater}=th}(\mathit{Sales}))|\}$$

and the second as

$$\{\langle th, ti, s\rangle \mid \langle th, ti\rangle \text{ is a theater-title pair appearing in } \mathit{Sales} \text{ and } s \text{ is the sum that includes each occurrence of each } a\text{-value in } \sigma_{\mathit{Theater}=th \wedge \mathit{Title}=ti}(\mathit{Sales})\}$$

A subtlety here is that this second query cannot be expressed simply as

$$\{\langle th, ti, s\rangle \mid \langle th, ti\rangle \text{ is a theater-title pair appearing in } \mathit{Sales} \text{ and } s = \Sigma\{a \in \pi_{\mathit{Attendance}}(\sigma_{\mathit{Theater}=th \wedge \mathit{Title}=ti}(\mathit{Sales}))\}\}$$

since a value a has to be counted as many times as it occurs in the selection. This suggests that a more natural setting for studying aggregate functions would explicitly include *bags* (or multisets, i.e., collections in which duplicates are permitted) and not just sets, a somewhat radical departure from the model we have used so far.

The two queries can be expressed as follows using aggregate functions in an algebraic language:

$$\pi_{\mathit{Theater};\ \mathbf{count}(\mathit{Title})}(\mathit{Sales})$$
$$\pi_{\mathit{Theater},\mathit{Title};\ \mathbf{sum}(\mathit{Attendance})}(\mathit{Sales}).$$

We now briefly present a more formal development. To simplify, the formalism is based on the unnamed perspective, and we assume that $\mathbf{dom} = \mathbf{N}$, i.e., the set of non-negative integers. We stay within the relational model although as noted in the preceding example, a richer data model with bags would be more natural. Indeed, the complex value model that will be studied in Chapter 20 provides a more appropriate context for considering aggregate functions.

We shall adopt a somewhat abstract view of aggregate operators. An *aggregate function* f is defined to be a family of functions $f_1, f_2, \ldots$ such that for each $j \geq 1$ and each relation schema S with $\mathit{arity}(S) \geq j$, $f_j : \mathit{Inst}(S) \to \mathbf{N}$. For instance, for the **sum** aggregate function, we will have $\mathbf{sum}_1$ to sum the first column and, in general, $\mathbf{sum}_i$ to sum the i^{th} one. As in the case of **sum**, we want the f_i to depend only on the content of the column to which they are applied, where the "content" includes not only the set of elements in the column, but also the number of their occurrences (so, columns are viewed as bags). This requirement is captured by the following *uniformity property* imposed on each aggregate function f:

Suppose that the i^{th} column of I and the j^{th} of J are identical, i.e., for each a, there are as many occurrences of a in the i^{th} column of I and in the j^{th} column of J. Then $f_i(I) = f_j(J)$.

All of the commonly arising aggregate functions satisfy this uniformity property. The uniformity condition is also used when translating calculus queries with aggregates into the algebra with aggregates.

We next illustrate how aggregate functions can be incorporated into the algebra and calculus (we do not discuss how this is done for nr-datalog$^{\neg}$, since it is similar to the algebra.) Aggregate functions are added to the algebra using an extended projection operation. Specifically, the projection function for aggregate function f on relation instance I is defined as follows:

$$\pi_{j_1,\ldots,j_m;f(k)}(I) = \{\langle a_{j_1}, \ldots, a_{j_m}, f_k(\sigma_{j_1=a_{j_1}\wedge\cdots\wedge j_m=a_{j_m}}(I))\rangle \mid \langle a_1, \ldots, a_n\rangle \in I\}.$$

Note that the aggregate function f_k is applied separately to each group of tuples in I corresponding to a different possible value for the columns $j_1, \ldots, j_m$.

Turning to the calculus, we begin with an example. Query (5.5) can be expressed in the extended calculus as

$$\begin{aligned}\{th, ti, s \mid \exists d_1, a_1(&Sales(th, ti, d_1, a_1)\\ &\wedge s = \mathbf{sum}_2\{d_2, a_2 \mid Sales(th, ti, d_2, a_2)\})\}\end{aligned}$$

where $\mathbf{sum}_2$ is the aggregate function summing the second column of a relation. Note that the subexpression $\{d_2, a_2 \mid Sales(th, ti, d_2, a_2)\}$ has free variables *th* and *ti* that do not occur in the target of the subexpression. Intuitively, different assignments for these variables will yield different values for the subexpression.

More formally, aggregate functions are incorporated into the calculus by permitting *aggregate terms* that have the form $f_j\{\vec{x} \mid \psi\}$, where f is an aggregate function, $j \leq arity(\vec{x})$ and ψ is a calculus formula (possibly with aggregate terms). When defining the extended calculus, care must be taken to guarantee that aggregate terms do not recursively depend on each other. This can be accomplished with a suitable generalization of safe range. This generalization will also ensure that free variables occurring in an aggregate term are range restricted by a subformula containing it. It is straightforward to define the semantics of the generalized safe-range calculus with aggregate functions. One can then show that the extensions of the algebra and safe-range calculus with the same set of aggregate functions have the same expressive power.

5.6 Digression: Finite Representations of Infinite Databases

Until now we have considered only finite instances of relational databases. As we have seen, this introduced significant difficulty in connection with domain independence of calculus queries. It is also restrictive in connection with some application areas that involve temporal or geometric data. For example, it would be convenient to think of a rectangle in the real plane as an infinite set of points, even if it can be represented easily in some finite manner.

In this short section we briefly describe some recent and advanced material that uses logic to permit the finite representation of infinite databases. We begin by presenting an alternative approach to resolving the problem of safety, that permits queries to have answers that are infinite but finitely representable. We then introduce a promising generalization of the relational model that uses constraints to represent infinite databases, and we describe how query processing can be performed against these in an efficient manner.

An Alternative Resolution to the Problem of Safety

As indicated earlier, much of the research on safety has been directed at syntactic restrictions to ensure domain independence. An alternative approach is to use the natural interpretation, even if the resulting answer is infinite. As it turns out, the answers to such queries are recursive and have a finite representation.

For this result, we shall use a finite set $\mathbf{d} \subset \mathbf{dom}$, which corresponds intuitively to the active domain of a query and input database; and a set $C = \{c_1, \ldots, c_m\}$ of m distinct "new" symbols, which will serve as placeholders for elements of $\mathbf{dom} - \mathbf{d}$. Speaking intuitively, the elements of C sometimes act as elements of **dom**, and so it is not appropriate to view them as simple variables.

A *tuple with placeholders* is a tuple $t = \langle t_1, \ldots, t_n \rangle$, where each t_i is in $\mathbf{d} \cup C$. The *semantics* of such t relative to $\mathbf{d}$ are

$$\begin{aligned} sem_{\mathbf{d}}(t) = \{\rho(t) \mid \rho \text{ is a one-one mapping from } \mathbf{d} \cup C \\ \text{that leaves } \mathbf{d} \text{ fixed and maps } C \text{ into } \mathbf{dom} - \mathbf{d}\}. \end{aligned}$$

The following theorem, stated without proof, characterizes the result of applying an arbitrary calculus query using the natural semantics.

Theorem 5.6.1 Let $q = \{e_1, \ldots, e_n \mid \varphi\}$ be an arbitrary calculus query, such that each quantifier in φ quantifies a distinct variable that is not free in φ. Let $C = \{c_1, \ldots, c_m\}$ be a set of m distinct "new" symbols not occurring in **dom**, but viewed as domain elements, where m is the number of distinct variables in φ. Then for each input instance **I**,

$$q_{\mathbf{dom}}(\mathbf{I}) = \cup\{sem_{adom(q,\mathbf{I})}(t) \mid t \in q_{adom(q,\mathbf{I}) \cup C}(\mathbf{I})\}.$$

This shows that if we apply a calculus query (under the natural semantics) to a finite database, then the result is recursive, even if infinite. But is the set of infinite databases described in this manner closed under the application of calculus queries? The affirmative answer is provided by an elegant generalization of the relational model presented next (see Exercise 5.31).

Constraint Query Languages

The following generalization of the relational model seems useful to a variety of new applications. The starting point is to consider infinite databases with finite representations based on the use of constraints. To begin we define a generalized n-tuple as a conjunction of constraints over n variables. The constraints typically include $=$, $\neq$, $\leq$, etc. In some sense, such a constraint can be viewed as a finite representation of a (possibly infinite) set of (normal) n-tuples (i.e., the valuations of the variables that satisfy the constraint).

Example 5.6.2 Consider the representation of rectangles in the plane. Suppose first that rectangles are given using 5-tuples (n, x_1, y_1, x_2, y_2), where n is the name of the rectangle, (x_1, y_1) are the coordinates of the lower left corner, and (x_2, y_2) are the coordinates of the upper right. The set of points $\langle u, v\rangle$ in such a rectangle delimited by x_1, y_1, x_2, y_2 is given by the constraint

$$x_1 \leq u \leq x_2 \wedge y_1 \leq v \leq y_2.$$

Now the names of intersecting rectangles from a relation R are given by

$$\begin{aligned}\{\langle n_1, n_2\rangle \mid \exists\ & x_1, y_1, x_2, y_2, x_1', y_1', x_2', y_2', u, v \\ & (R(n_1, x_1, y_1, x_2, y_2) \wedge (x_1 \leq u \leq x_2 \wedge y_1 \leq v \leq y_2)\wedge \\ & R(n_2, x_1', y_1', x_2', y_2') \wedge (x_1' \leq u \leq x_2' \wedge y_1' \leq v \leq y_2'))\}.\end{aligned}$$

This is essentially within the framework of the relational model presented so far, except that we are using an infinite base relation $\leq$. There is a level of indirection between the representation of a rectangle (a, x_1, y_1, x_2, y_2) and the actual set of points that it contains.

In the following constraint formalism, a named rectangle can be represented by a "generalized tuple" (i.e., a constraint). For instance, the rectangle of name a with corners (0.5, 1.0) and (1.5, 5.5) is represented by the constraint

$$z_1 = a \wedge 0.5 \leq z_2 \wedge z_2 \leq 1.5 \wedge 1.0 \leq z_3 \wedge z_3 \leq 5.5.$$

This should be viewed as a finite syntactic representation of an infinite set of triples. A triple $\langle z_1, z_2, z_3\rangle$ satisfying this constraint indicates that the point of coordinates (z_2, z_3) is in a rectangle with name z_1.

One can see a number of uses in allowing constraints in the language. First, constraints arise naturally for domains concerning measures (price, distance, time, etc.). The introduction of time has already been studied in the active area of temporal databases (see Section 22.6). In other applications such as spatial databases, geometry plays an essential role and fits nicely in the realm of constraint query languages.

One can clearly obtain different languages by considering various domains and various forms of constraints. Relational calculus, relational algebra, or some other relational languages can be extended with, for instance, the theory of real closed fields or the theory of dense orders without endpoints. Of course, a requirement is the decidability of the resulting language.

Assuming some notion of constraints (to be formalized soon), we now define somewhat more precisely the constraint languages and then illustrate them with two examples.

Definition 5.6.3 A *generalized n-tuple* is a finite conjunction of constraints over variables $x_1, \ldots, x_n$. A *generalized instance* of arity n is a finite set of generalized n-tuples (the corresponding formula is the disjunction of the constraints).

Suppose that I is a generalized instance. We refer to I as a *syntactic* database and to the set of conventional tuples represented by I as the *semantic* database.

We now present two applications of this approach, one in connection with the reals and the other with the rationals.

We assume now that the constants are interpreted over a real closed field (e.g., the reals). The constraints are polynomial inequality constraints [i.e., inequalities of the form $p(x_1, \ldots, x_n) \geq 0$, where p is a polynomial]. Two 3-tuples in this context are

$$(3.56 \times x_1^2 + 4.0 \times x_2 \geq 0) \wedge (x_3 - x_1 \geq 0)$$
$$(x_1 + x_2 + x_3 \geq 0).$$

One can evaluate queries algebraically bottom-up (i.e., at each step of the computation, the result is still a generalized instance). This is a straightforward consequence of Tarski's decision procedure for the theory of real closed fields. A difficulty resides in projection (i.e., quantifier elimination). The procedure for projection is extremely costly in the size of the query. However, for a fixed query, the complexity *in the size of the syntactic database* is reasonable (in NC).

We assume now that the constants are interpreted over a countably infinite set with a binary relation $\leq$ that is a dense order (e.g., the rationals). The constraints are of the form $x\theta y$ or $x\theta c$, where x, y are variables, c is a constant, and θ is among $\leq, <, =$. An example of a 3-tuple is

$$(x_1 \leq x_2) \wedge (x_2 < x_3).$$

Here again, a bottom-up algebraic evaluation is feasible. Indeed, evaluation is in AC_0 in the size of the syntactic database (for a fixed query).

In the remainder of this book, we consider standard databases and not generalized ones.

Bibliographic Notes

One of the first investigations of using predicate calculus to query relational database structures is [Kuh67], although the work by Codd [Cod70, Cod72b] played a tremendous role in bringing attention to the relational model and to the relational algebra and calculus. In particular, [Cod72b] introduced the equivalence of the calculus and algebra to the database community. That paper coined the phrase *relational completeness* to describe the expressive power of relational query languages: Any language able to simulate the algebra is called relationally complete. We have not emphasized that phrase here because subsequent research has suggested that a more natural notion of completeness can be described in terms of Turing computability (see Chapter 16).

Actually, a version of the result on the equivalence of the calculus and algebra was known much earlier to the logic community (see [CT48, TT52]) and is used to show Tarski's algebraization theorem (e.g., see [HMT71]). The relation between relational algebras and cylindric algebras is studied in [IL84] (see Exercise 5.36). The development of algebras equivalent to various calculus languages has been a fertile area for database theory. One such result presented in this chapter is the equivalence of the positive existential calculus and the SPCU algebra [CH82]; analogous results have also been developed for the relational calculus extended with aggregate operators [Klu82], the complex value model [AB88] (studied in Chapter 20), the Logical Data Model [KV84], the directory model [DM86a, DM92], and formalizations of the hierarchy and network models using database logic [Jac82].

Notions related to domain independence are found as early as [Low15] in the logic community; in the database community the first paper on this topic appears to be [Kuh67], which introduced the notion of *definite* queries. The notion of domain independence used here is from [Fag82b, Mak81]; the notions of definite and domain independent were proved equivalent in [ND82]. A large number of classes of domain-independent formulas have been investigated. These include the safe [Ull82b], safe DRC [Ull88], range separable [Cod72b], allowed [Top87] (Exercise 5.25), range restricted [Nic82] (Exercise 5.26), and evaluable [Dem82] formulas. An additional investigation of domain independence for the calculus is found in [ND82]. Surveys on domain independence and various examples of these classes can be found in [Kif88, VanGT91]. The focus of [VanGT91] is the general problem of practical translations from calculus to algebra queries; in particular, it provides a translation from the set of evaluable formulas into the algebra. It is also shown there that the notions of evaluable and range restricted are equivalent. These are the most general syntactic conditions in the literature that ensure domain independence. The fact that domain independence is undecidable was first observed in [DiP69]; this is detailed in Chapter 6.

Domain independence also arises in the context of dependencies [Fag82b, Mak81] and datalog [Dec86, Top87, RBS87, TS88]. The issue of extending domain independence to incorporate functions (e.g., arithmetic functions, or user-defined functions) is considered in [AB88, Top91, EHJ93]. The issue of extending domain independence to incorporate freely interpreted functions (such as arise in logic programming) is addressed in [Kif88]. Syntactic conditions on (recursive) datalog programs with arithmetic that ensure safety are developed in [RBS87, KRS88a, KRS88b, SV89]. Issues of safety in the presence of function or order symbols are also considered in [AH91]. Aggregate functions were first incorporated into the relational algebra and calculus in [Klu82]; see also [AB88].

The notion of safe range presented here is richer than safe, safe DRC, and range separable and weaker than allowed, evaluable, and range restricted. It follows the spirit of the definition of allowed presented in [VanGT91] and safe range in [AB88]. The transformations of the safe-range calculus to the algebra presented here follows the more general transformations in [VanGT91, EHJ93]. The notion of "relational algebra normal form" used in those works is more general than the notion by that name used here.

Query languages have mostly been considered for finite databases. An exception is [HH93]. Theorem 5.6.1 is due to [AGSS86]. An alternative proof and extension of this result is developed in [HS94].

Programming with constraints has been studied for some time in topic areas ranging from linear programming to AI to logic programming. Although the declarative spirit of both constraint programming and database query languages leads to a natural marriage, it is only recently that the combination of the two paradigms has been studied seriously [KKR90]. This was probably a consequence of the success of constraints in the field of logic programming (see, e.g., [JL87] and [Lel87, Coh90] for surveys). Our presentation was influenced by [KKR90] (calculii for closed real fields with polynomial inequalities and for dense order with inequalities are studied there) as well as by [KG94] (an algebra for constraint databases with dense order and inequalities is featured there). Recent works on constraint database languages can be found in [Kup93, GS94].

Exercises

Exercise 5.1 Express queries (5.2 and 5.3) in (1) the relational algebras, (2) nonrecursive datalog$^\neg$, and (3) domain-independent relational calculus.

Exercise 5.2 Express the following queries against the *CINEMA* database in (1) the relational algebras, (2) nonrecursive datalog$^\neg$, and (3) domain-independent relational calculus.

(a) Find the actors cast in at least one movie by Kurosawa.
(b) Find the actors cast in every movie by Kurosawa.
(c) Find the actors cast only in movies by Kurosawa.
(d) Find all pairs of actors who act together in at least one movie.
(e) Find all pairs of actors cast in exactly the same movies.
(f) Find the directors such that every actor is cast in one of his or her films.

(Assume that each film has exactly one director.)

Exercise 5.3 Prove or disprove (assuming $X \subseteq sort(P) = sort(Q)$):

(a) $\pi_X(P \cup Q) = \pi_X(P) \cup \pi_X(Q)$;
(b) $\pi_X(P \cap Q) = \pi_X(P) \cap \pi_X(Q)$.

Exercise 5.4

(a) Give formal definitions for the syntax and semantics of the unnamed and named relational algebras.
(b) Show that in the unnamed algebra $\cap$ can be simulated using (1) the difference operator $-$; (2) the operators $\times, \pi, \sigma$.
(c) Give a formal definition for the syntax and semantics of selection operators in the unnamed algebra that permit conjunction, disjunction, and negation in their formulas. Show that these selection operators can be simulated using atomic selection operators, union, intersect, and difference.
★ (d) Show that the SPCU algebra, in which selection operators with negation in the formulas are permitted, cannot simulate the difference operator.
★ (e) Formulate and prove results analogous to those of parts (b), (c), and (d) for the named algebra.

Exercise 5.5

(a) Prove that the unnamed algebra operators $\{\sigma, \pi, \times, \cup, -\}$ are nonredundant.
(b) State and prove the analogous result for the named algebra.

Exercise 5.6

(a) Exhibit a relational algebra query that is not monotonic.
(b) Exhibit a relational algebra query that is not satisfiable.

Exercise 5.7 Prove Proposition 5.1.2 (i.e., that the unnamed and named relational algebras have equivalent expressive power).

Exercise 5.8 (Division) The *division* operator, denoted $\div$, is added to the named algebra as follows. For instances I and J with $sort(J) \subseteq sort(I)$, the value of $I \div J$ is the set of tuples $r \in \pi_{sort(I)-sort(J)}(I)$ such that $(\{r\} \bowtie J) \subseteq I$. Use the division to express algebraically the query, "Which theater is featuring all of Hitchcock's movies?". Describe how nr-datalog$^\neg$ can be used to simulate division. Describe how the named algebra can simulate division. Is division a monotonic operation?

Exercise 5.9 Show that the semantics of each nr-datalog$^\neg$ rule can be described as a difference $q_1 - q_2$, where q_1 is an SPJR query and q_2 is an SPJRU query.

Exercise 5.10 Verify that each nr-datalog$^\neg$ program with equality can be simulated by one without equality.

Exercise 5.11 Prove Proposition 5.2.2. *Hint:* Use the proof of Theorem 4.4.8 and the fact that the relational algebra is closed under composition.

★ **Exercise 5.12** Prove that the domain-independent relational calculus without equality is strictly weaker than the domain-independent relational calculus. *Hint:* Suppose that calculus query q without equality is equivalent to $\{x \mid R(x) \wedge x \neq a\}$. Show that q can be translated into an algebra query q' that is constructed without using a constant base relation and such that all selections are on base relation expressions. Argue that on each input relation I over R, each subexpression of q' evaluates to either I^n for some $n \geq 0$, or to the empty relation for some $n \geq 0$.

Exercise 5.13

(a) Complete the proof of Lemma 5.3.11.

(b) Complete the proof of Lemma 5.3.12.

Exercise 5.14

(a) Prove that the rewrite rules of Figure 5.1 preserve equivalence.

(b) Prove that these rewrite rules can be used to transform an arbitrary calculus formula into an equivalent formula in PNF with CNF matrix. State which rewrite rules are needed.

(c) Do the same as (b), but for DNF matrix.

(d) Prove that the rewrite rules of Figure 5.1 are not complete in the sense that there are calculus formulas φ and ψ such that (1) $\varphi \equiv \psi$, but (2) there is no sequence of applications of the rewrite rules that transforms φ into ψ.

Exercise 5.15 Verify the claims of Example 5.3.9.

Exercise 5.16

(a) Show that each positive existential query is equivalent to one whose formula is in PNF with either CNF or DNF matrix and that they can be expressed in the form $\{e_1, \ldots, e_n \mid \psi_1 \vee \cdots \vee \psi_m\}$, where each ψ_j is a conjunctive calculus formula with $free(\psi_j)$ = the set of variables occurring in $e_1, \ldots, e_n$. Note that this formula is safe range.

(b) Show that it is decidable, given a relational calculus query q (possibly with equality) whose only logical connectives are $\wedge$, $\vee$, and $\exists$, whether q is domain independent.

(c) Prove Theorem 5.4.10.

Exercise 5.17 Use the construction of the proof of Theorem 5.4.6 to transform the following into the algebra.

(a) $\{\langle\,\rangle \mid \exists x(R(x) \wedge \exists y(S(x, y) \wedge \neg\exists z(T(x, y, a))))\}$

(b) $\{w, x, y, z \mid (R(w, x, y) \vee R(w, x, z)) \wedge (R(y, z, w) \vee R(y, z, x))\}$

Exercise 5.18 For each of the following queries, indicate whether it is domain independent and/or safe range. If it is not domain independent, give examples of different domains yielding different answers on the same input; and if it is safe range, translate it into the algebra.

(a) $\{x, y \mid \exists z[T(x, z) \wedge \exists w T(w, x, y)] \wedge x = y\}$

(b) $\{x, y \mid [x = a \vee \exists z(R(y, z))] \wedge S(y)\}$

(c) $\{x, y \mid [x = a \vee \exists z(R(y, z))] \wedge S(y) \wedge T(x)\}$

(d) $\{x \mid \forall y(R(y) \rightarrow S(x, y))\}$

(e) $\{\langle\rangle \mid \exists x\forall y(R(y) \rightarrow S(x, y))\}$

(f) $\{x, y \mid \exists z T(x, y, z) \wedge \exists u, v([(R(u) \vee S(u, v)) \wedge R(v)]$
$\rightarrow [\exists w(T(x, w, v) \wedge T(u, v, y))])\}$

★ **Exercise 5.19** Consider the proof of Theorem 5.4.6.

(a) Give the missing parts of Algorithm 5.4.7.

(b) Show that Algorithm 5.4.7 is correct and terminates on all input.

(c) Give the missing parts of Algorithm 5.4.8 and verify its correctness.

(d) Given $q = \{\langle x_1, \ldots, x_n\rangle \mid \varphi\}$ with φ in modified RANF, show for each instance $\mathbf{I}$ and each $\mathbf{d}$ satisfying $adom(q, \mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$ that $q_{\mathbf{d}}(\mathbf{I}) = E_\varphi(\mathbf{I})$.

Exercise 5.20 Consider the proof of Theorem 5.4.6.

(a) Present examples illustrating how the nondeterministic choices in these rewrite rules can be used to help optimize the algebra query finally produced by the construction of the proof of this lemma. (Refer to Chapter 6 for a general discussion of optimization.)

(b) Consider a generalization of rules (R1) and (R2) that permits using a set of indexes $\{j_1, \ldots, j_l\} \subseteq \{1, \ldots, n\} - \{i_1, \ldots, i_k\}$. What are the advantages of this generalization? What restrictions must be imposed to ensure that Algorithm 5.4.8 remains correct?

Exercise 5.21 Develop a direct proof that $\text{CALC}_{adom} \sqsubseteq \text{CALC}_{sr}$. *Hint:* Given calculus query q, first build a formula $\xi_{adom}(x)$ such that $\mathbf{I} \models \xi_{adom}(x)[\nu]$ iff $\nu(x) \in adom(q, \mathbf{I})$. Now perform an induction on subformulas.

★ **Exercise 5.22** [Coh86] Let R have arity n. Define the *gen*(*erator*) operator so that for instance I of R, indexes $1 \leq i_1 < \cdots < i_k \leq n$, and constants $a_1, \ldots, a_k$,

$$gen_{i_1:a_1,\ldots,i_k:a_k}(I) = \pi_{j_1,\ldots,j_l}(\sigma_{i_1=a_1\wedge\cdots\wedge i_k=a_k}(I)),$$

where $\{j_1, \ldots, j_l\}$ is a listing in order of (some or all) indexes in $\{1, \ldots, n\} - \{i_1, \ldots, i_k\}$. Note that the special case of $gen_{1:b_1,\ldots,n:b_n}(I)$ can be viewed as a test of $\langle b_1, \ldots, b_n\rangle \in I$; and $gen_{[\,]}(I)$

is a test of whether I is nonempty. In some research in AI, the primitive mechanism for accessing relations is based on generators that are viewed as producing a stream of tuples as output. For example, the query $\{\langle x, y, z\rangle \mid R(x, y) \wedge S(y, z)\}$ can be computed using the algorithm

```
for each tuple $\langle x, y\rangle$ generated by $gen_{1:x,2:y}(R)$
  for each value $\langle z\rangle$ generated by $gen_{1:y}(S)$
    output $\langle x, y, z\rangle$
  end for each
end for each
```

Develop an algorithm for translating calculus queries into programs using generators. Describe syntactic restrictions on the calculus that ensure that your algorithm succeeds.

♠**Exercise 5.23** [Cod72b] (Tuple calculus.) We use a set **tvar** of sorted tuple variables. The *tuple calculus* is defined as follows. If t is a tuple variable and A is an attribute in the sort of t, $t.A$ is a *term*. A constant is also a *term*. The atomic formulas are either of the form $R(t)$ with the appropriate constraint on sorts, or $e = e'$, where e, e' are terms. Formulas are constructed as in the standard relational calculus. For example, query (5.1) is expressed by the tuple calculus query

$$\begin{aligned}\{t\colon title \mid \exists s\colon title, director, actor[&Movie(s) \wedge t.title = s.title \\ &\wedge s.director = \text{“Hitchcock”}] \\ \wedge \neg\exists u\colon title, director, actor[&Movie(u) \wedge u.title = s.title \\ &\wedge u.actor = \text{“Hitchcock”}]\}.\end{aligned}$$

Give a formal definition for the syntax of the tuple calculus and for the relativized interpretation, active domain, and domain-independent semantics. Develop an analog of safe range. Prove the equivalence of conventional calculus and tuple calculus under all of these semantics.

Exercise 5.24 Prove that the relational calculus and the family of nr-datalog$^\neg$ programs with single-relation output have equivalent expressive power by using direct simulations between the two families.

♠**Exercise 5.25** [Top87] Let **R** be a database schema, and define the binary relation *gen*(*erates*) on variables and formulas as follows:

$gen(x, \varphi)$	if $\varphi = R(u)$ for some $R \in \mathbf{R}$ and $x \in free(\varphi)$
$gen(x, \neg\varphi)$	if $gen(x, pushnot(\neg\varphi))$
$gen(x, \exists y\varphi)$	if x, y are distinct and $gen(x, \varphi)$
$gen(x, \forall y\varphi)$	if x, y are distinct and $gen(x, \varphi)$
$gen(x, \varphi \vee \psi)$	if $gen(x, \varphi)$ and $gen(x, \psi)$
$gen(x, \varphi \wedge \psi)$	if $gen(x, \varphi)$ or $gen(x, \psi)$,

where $pushnot(\neg\varphi)$ is defined in the natural manner to be the result of pushing the negation into the next highest level logical connective (with consecutive negations cancelling each other) unless φ is an atom (using the rewrite rules 5, 6, 7, 10, and 11 from Fig. 5.1). A formula φ is *allowed*

(i) if $x \in free(\varphi)$ then $gen(x, \varphi)$;

(ii) if for each subformula $\exists y\psi$ of φ, $gen(y, \psi)$ holds; and

(iii) if for each subformula $\forall y \psi$ of φ, $gen(y, \neg\psi)$ holds.

A calculus query is *allowed* if its formula is allowed.

(a) Exhibit a query that is allowed but not safe range.

★ (b) Prove that each allowed query is domain independent.

In [VanGT91, EHJ93] a translation of allowed formulas into the algebra is presented.)

★ **Exercise 5.26** [Nic82] The notion of "range-restricted" queries, which ensures domain independence, is based on properties of the normal form equivalents of queries. Let $q = \{\vec{x} \mid \varphi\}$ be a calculus query, and let $\varphi_{DNF} = \vec{\%}y(D_1 \vee \cdots \vee D_n)$ be the result of transforming φ into PNF with DNF matrix using the rewrite rules of Fig. 5.1; and similarly let $\varphi_{CNF} = \vec{\%}z(C_1 \wedge \cdots \wedge C_m)$ be the result of transforming φ into PNF with CNF matrix. The query q is *range restricted* if

(i) each free variable x in φ occurs in a positive literal (other than $x = y$) in every D_i;

(ii) each existentially quantified variable x in φ_{DNF} occurs in a positive literal (other than $x = y$) in every D_i where x occurs; and

(iii) each universally quantified variable x in φ_{CNF} occurs in a negative literal (other than $x \neq y$) in every C_j where x occurs.

Prove that range-restricted queries are domain independent. (In [VanGT91] a translation of the range-restricted queries into the algebra is presented.)

Exercise 5.27 [VanGT91] Suppose that $R[Product, Part]$ holds project numbers and the parts that are used to make them, and $S[Supplier, Part]$ holds supplier names and the parts that they supply. Consider the queries

$$q_1 = \{x \mid \forall y(R(100, y) \to S(x, y))\}$$
$$q_2 = \{\langle\rangle \mid \exists x \forall y(R(100, y) \to S(x, y))\}$$

(a) Prove that q_1 is not domain independent.

(b) Prove that q_2 is not allowed (Exercise 5.25) but it is range restricted (Exercise 5.26) and hence domain independent.

(c) Find an algebra query q' equivalent to q_2.

Exercise 5.28 [Klu82] Consider a database schema with relations *Dept*[*Name*, *Head*, *College*], *Faculty*[*Name*, *Dname*], and *Grad*[*Name*, *MajorProf*, *GrantAmt*], and the query

For each department in the Letters and Science College, compute the total graduate student support for each of the department's faculty members, and produce as output a relation that includes all pairs $\langle d, a\rangle$ where d is a department in the Letters and Science College, and a is the average graduate student support per faculty member in d.

Write algebra and calculus queries that express this query.

Exercise 5.29 We consider constraint databases involving polynomial inequalities over the reals. Let $I_1 = \{(9x_1^2 + 4x_2 \geq 0)\}$ be a generalized instance over AB, where x_1 ranges over A and x_2 ranges over B, and let $I_2 = \{(x_3 - x_1 \geq 0)\}$ over AC. Express $\pi_{BC}(I_1 \bowtie I_2)$ as a generalized instance.

★ **Exercise 5.30** Recall Theorem 5.6.1.

(a) Let finite $\mathbf{d} \subset \mathbf{dom}$ be fixed, C be a set of new symbols, and t be a tuple with placeholders. Describe a generalized tuple (in the sense of constraint databases) t' whose semantics are equal to $sem_{\mathbf{d}}(t)$.

(b) Show that the family of databases representable by sets of tuples with placeholders is closed under the relational calculus.

♠ **Exercise 5.31** Prove Theorem 5.6.1.

Exercise 5.32 [Mai80] (Unrestricted algebra) For this exercise we permit relations to be finite or infinite. Consider the *complement* operator ^{c} defined on instances I of arity n by $I^c = \mathbf{dom}^n - I$. (The analogous operator is defined for the named algebra.) Prove that the calculus under the natural interpretation is equivalent to the algebra with operators $\{\sigma, \pi, \times, \cup, ^c\}$.

★ **Exercise 5.33** A total mapping τ from instances over $\mathbf{R}$ to instances over S is C-*generic* for $C \subseteq \mathbf{dom}$, iff for each bijection ρ over $\mathbf{dom}$ that is the identity on C, τ and ρ commute. That is, $\tau(\rho(\mathbf{I})) = \rho(\tau(\mathbf{I}))$ for each instance $\mathbf{I}$ of $\mathbf{R}$. The mapping τ is *generic* if it is C-generic for some finite $C \subseteq \mathbf{dom}$. Prove that each relational algebra query is generic—in particular, that each algebra query q is $adom(q)$-generic.

♠ **Exercise 5.34** Let R be a unary relation name. A *hyperplane query* over R is a query of the form $\sigma_F(R \times \cdots \times R)$ (with 0 or more occurrences of R), where F is a conjunction of atoms of the form $i = j$, $i \neq j$, $i = a$, or $i \neq a$ (for indexes i, j and constant a). A formula F of this form is called a *hyperplane formula*. A *hyperplane-union query* over R is a query of the form $\sigma_F(R \times \cdots \times R)$, where F is a disjunction of hyperplane formulas; a formula of this form is called a *hyperplane-union* formula.

(a) Show that if q is an algebra query over R, then there is an $n \geq 0$ and a hyperplane-union query q' such that for all instances I over R, if $|I| \geq n$ and $adom(I) \cap adom(q) = \emptyset$, then $q(I) = q'(I)$.

The query *even* is defined over R as follows: $even(I) = \{\langle\rangle\}$ (i.e., yes) if $|I|$ is even; and $even(I) = \{\}$ (i.e., no) otherwise.

(b) Prove that there is no algebra query q over R such that $q \equiv even$.

Exercise 5.35 [CH80b] (Unsorted algebra) An *h-relation* (for heterogeneous relation) is a finite set of tuples not necessarily of the same arity.

(a) Design an algebra for h-relations that is at least as expressive as relational algebra.

★ (b) Show that the algebra in (a) can be chosen to have the additional property that if q is a query in this algebra taking standard relational input and producing standard relational output, then there is a standard algebra query q' such that $q' \equiv q$.

♠ **Exercise 5.36** [IL84] (Cylindric algebra) Let n be a positive integer, $R[A_1, \ldots, A_n]$ a relation schema, and C a (possibly infinite) set of constants. Recall that a *Boolean algebra* is a 6-tuple $(\mathcal{B}, \vee, \wedge, \bar{\ }, \bot, \top)$, where $\mathcal{B}$ is a set containing $\bot$ and $\top$; $\vee, \wedge$ are binary operations on $\mathcal{B}$; and $\bar{\ }$ is a unary operation on $\mathcal{B}$ such that for all $x, y, z \in \mathcal{B}$:

(a) $x \vee y = y \vee x$;

(b) $x \wedge y = y \wedge x$;

(c) $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$;

(d) $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$;

(e) $x \wedge \bot = \bot$;
(f) $x \vee \top = \top$;
(g) $x \wedge \overline{x} = \bot$;
(h) $x \vee \overline{x} = \top$; and
(i) $\bot \neq \top$.

For a Boolean algebra, define $x \leq y$ to mean $x \wedge y = x$.

(a) Show that $\langle \mathcal{R}_C, \cup, \cap, ^c, \emptyset, C^n \rangle$ is a Boolean algebra where $\mathcal{R}_C$ is the set of all (possibly infinite) R-relations over constants in C and ^{c} denotes the unary *complement* operator, defined so that $I^c = C^n - I$. In addition, show that $I \leq J$ iff $I \subseteq J$.

Let the diagonals d_{ij} be defined by the statement, "for each i, j, $d_{ij} = \sigma_{A_i = A_j}(C^n)$"; and let the i^{th} cylinder C_i be defined for each I by the statement, "$C_i I$ is the relation over $\mathcal{R}_C$ defined by

$$C_i I = \{t \mid \pi_{A_1 \ldots A_{i-1} A_{i+1} \ldots A_n}(t) \in \pi_{A_1 \ldots A_{i-1} A_{i+1} \ldots A_n}(I) \text{ and } t(A_i) \in C\}."$$

(b) Show the following properties of cylindric algebras: (1) $C_i \emptyset = \emptyset$; (2) $x \leq C_i x$; (3) $C_i(x \cap C_i y) = C_i x \cap C_i y$; (4) $C_i C_j x = C_j C_i x$; (5) $d_{ii} = C^n$; (6) if $i \neq j$ and $i \neq k$, then $d_{jk} = C_i(d_{ji} \cap d_{ik})$; (7) if $i \neq j$, then $C_i(d_{ij} \cap x) \cap C_i(d_{ij} \cap \overline{x}) = \emptyset$.

(c) Let h be the mapping from any (possibly infinite) relation S with $sort(S) \subset A_1 \ldots A_n$ with entries in C to a relation over R obtained by extending each tuple in S to $A_1 \ldots A_n$ in all possible ways with values in C. Prove that (1) $h(R_1 \bowtie R_2) = h(R_1) \cap h(R_2)$ and (2) if $A_1 \in sort(R)$, then $h(\pi_{A_1}(R)) = C_1 h(R_1)$.