Propositional Logic

We begin with the *propositional calculus*. For this we assume an infinite set of *propositional variables*, typically denoted $p, q, r, \ldots$, possibly with subscripts. We also permit the special *propositional constants true* and *false*. (*Well-formed*) *propositional formulas* are constructed from the propositional variables and constants, using the unary connective *negation* ($\neg$) and the binary connectives *disjunction* ($\vee$), *conjunction* ($\wedge$), *implication* ($\rightarrow$), and *equivalence* ($\leftrightarrow$). For example, p, $(p \wedge (\neg q))$ and $((p \vee q) \rightarrow p)$ are well-formed propositional formulas. We generally omit parentheses if not needed for understanding a formula.

A *truth assignment* for a set V of propositional variables is a function $\xi : V \rightarrow \{true, false\}$. The *truth value* $\varphi[\xi]$ of a propositional formula φ under truth assignment ξ for the variables occurring in φ is defined by induction on the structure of φ in the natural manner. For example,

- $true[\xi] = true$;
- if $\varphi = p$ for some variable p, then $\varphi[\xi] = \xi(p)$;
- if $\varphi = (\neg\psi)$ then $\varphi[\xi] = true$ iff $\psi[\xi] = false$;
- $(\psi_1 \vee \psi_2)[\xi] = true$ iff at least one of $\psi_1[\xi] = true$ or $\psi_2[\xi] = true$.

If $\varphi[\xi] = true$ we say that $\varphi[\xi]$ is true and that φ is true under ξ (and similarly for false).

A formula φ is *satisfiable* if there is at least one truth assignment that makes it true, and it is *unsatisfiable* otherwise. It is *valid* if each truth assignment for the variables in φ makes it true. The formula $(p \vee q)$ is satisfiable but not valid; the formula $(p \wedge (\neg p))$ is unsatisfiable; and the formula $(p \vee (\neg p))$ is valid.

A formula φ *logically implies* formula ψ (or ψ is a *logical consequence of* φ), denoted $\varphi \models \psi$ if for each truth assignment ξ, if $\varphi[\xi]$ is true, then $\psi[\xi]$ is true. Formulas φ and ψ are (*logically*) *equivalent*, denoted $\varphi \equiv \psi$, if $\varphi \models \psi$ and $\psi \models \varphi$.

For example, $(p \wedge (p \rightarrow q)) \models q$. Many equivalences for propositional formulas are well known. For example,

$$(\varphi_1 \rightarrow \varphi_2) \equiv ((\neg\varphi_1) \vee \varphi_2); \quad \neg(\varphi_1 \vee \varphi_2) \equiv (\neg\varphi_1 \wedge \neg\varphi_2);$$
$$(\varphi_1 \vee \varphi_2) \wedge \varphi_3 \equiv (\varphi_1 \wedge \varphi_3) \vee (\varphi_2 \wedge \varphi_3); \quad \varphi_1 \wedge \neg\varphi_2 \equiv \varphi_1 \wedge (\varphi_1 \wedge \neg\varphi_2);$$
$$(\varphi_1 \vee (\varphi_2 \vee \varphi_3)) \equiv ((\varphi_1 \vee \varphi_2) \vee \varphi_3).$$

Observe that the last equivalence permits us to view $\vee$ as a polyadic connective. (The same holds for $\wedge$.)

A *literal* is a formula of the form p or $\neg p$ (or *true* or *false*) for some propositional variable p. A propositional formula is in *conjunctive normal form* (CNF) if it has the form $\psi_1 \wedge \cdots \wedge \psi_n$, where each formula ψ_i is a disjunction of literals. *Disjunctive normal form* (DNF) is defined analogously. It is known that if φ is a propositional formula, then there is some formula ψ equivalent to φ that is in CNF (respectively DNF). Note that if φ is in CNF (or DNF), then a shortest equivalent formula ψ in DNF (respectively CNF) may have a length exponential in the length of φ.

First-Order Logic

We now turn to *first-order predicate calculus*. We indicate the main intuitions and concepts underlying first-order logic and describe the primary specializations typically made for database theory. Precise definitions of needed portions of first-order logic are included in Chapters 4 and 5.

First-order logic generalizes propositional logic in several ways. Intuitively, propositional variables are replaced by predicate symbols that range over n-ary relations over an underlying set. Variables are used in first-order logic to range over elements of an abstract set, called the *universe of discourse*. This is realized using the quantifiers $\exists$ and $\forall$. In addition, function symbols are incorporated into the model. The most important definitions used to formalize first-order logic are first-order language, interpretation, logical implication, and provability.

Each first-order language L includes a set of variables, the propositional connectives, the quantifiers $\exists$ and $\forall$, and punctuation symbols ")", "(", and ",". The variation in first-order languages stems from the symbols they include to represent constants, predicates, and functions. More formally, a first-order language includes

(a) a (possibly empty) set of *constant* symbols;

(b) for each $n \geq 0$ a (possibly empty) set of *n-ary predicate symbols*;

(c) for each $n \geq 1$ a (possibly empty) set of *n-ary function symbols*.

In some cases, we also include

(d) the equality symbol $\approx$, which serves as a binary predicate symbol,

and the propositional constants *true* and *false*. It is common to focus on languages that are finite, except for the set of variables.

A familiar first-order language is the language $L_{\mathbf{N}}$ of the nonnegative integers, with

(a) constant symbol $\mathbf{0}$;

(b) binary predicate symbol $\leq$;

(c) binary function symbols $+$, $\times$, and unary $\mathbf{S}$ (successor);

and the equality symbol.

Let L be a first-order language. *Terms* of L are built in the natural fashion from constants, variables, and the function symbols. An *atom* is either *true*, *false*, or an expression of the form $R(t_1, \ldots, t_n)$, where R is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms. Atoms correspond to the propositional variables of propositional logic. If the equality symbol is included, then atoms include expressions of the form $t_1 \approx t_2$. The family of (*well-formed predicate calculus*) *formulas* over L is defined recursively starting with atoms, using the Boolean connectives, and using the quantifiers as follows: If φ is a formula and x a variable, then $(\exists x\varphi)$ and $(\forall x\varphi)$ are formulas. As with the propositional case, parentheses are omitted when understood from the context. In addition, $\vee$ and $\wedge$ are viewed as polyadic connectives. A term or formula is *ground* if it involves no variables.

Some examples of formulas in $L_{\mathbf{N}}$ are as follows:

$$\forall x(\mathbf{0} \leq x), \quad \neg(x \approx \mathbf{S}(x)),$$
$$\neg\exists x(\forall y(y \leq x)), \quad \forall y \forall z(x \approx y \times z \rightarrow (y \approx \mathbf{S}(\mathbf{0}) \vee z \approx \mathbf{S}(\mathbf{0}))).$$

(For some binary predicates and functions, we use infix notation.)

The notion of the scope of quantifiers and of *free* and *bound* occurrences of variables in formulas is now defined using recursion on the structure. Each variable occurrence in an atom is free. If φ is $(\psi_1 \vee \psi_2)$, then an occurrence of variable x in φ is free if it is free as an occurrence of ψ_1 or ψ_2; and this is extended to the other propositional connectives. If φ is $\exists y \psi$, then an occurrence of variable $x \neq y$ is free in φ if the corresponding occurrence is free in ψ. Each occurrence of y is bound in φ. In addition, each occurrence of y in φ that is free in ψ is said to be in the *scope* of $\exists y$ at the beginning of φ. A *sentence* is a well-formed formula that has no free variable occurrences.

Until now we have not given a meaning to the symbols of a first-order language and thereby to first-order formulas. This is accomplished with the notion of *interpretation*, which corresponds to the truth assignments of the propositional case. Each interpretation is just one of the many possible ways to give meaning to a language.

An *interpretation* of a first-order language L is a 4-tuple $\mathcal{I} = (U, \mathcal{C}, \mathcal{P}, \mathcal{F})$ where U is a nonempty set of abstract elements called the *universe* (*of discourse*), and $\mathcal{C}$, $\mathcal{P}$, and $\mathcal{F}$ give meanings to the sets of constant symbols, predicate symbols, and function symbols. For example, $\mathcal{C}$ is a function from the constant symbols into U, and $\mathcal{P}$ maps each n-ary predicate symbol p into an n-ary relation over U (i.e., a subset of U^n). It is possible for two distinct constant symbols to map to the same element of U.

When the *equality symbol* denoted $\approx$ is included, the meaning associated with it is restricted so that it enjoys properties usually associated with equality. Two equivalent mechanisms for accomplishing this are described next.

Let $\mathcal{I}$ be an interpretation for language L. As a notational shorthand, if c is a constant symbol in L, we use $c^{\mathcal{I}}$ to denote the element of the universe associated with c by $\mathcal{I}$. This is extended in the natural way to ground terms and atoms.

The usual interpretation for the language $L_{\mathbf{N}}$ is $\mathcal{I}_{\mathbf{N}}$, where the universe is $\mathbf{N}$; $\mathbf{0}$ is mapped to the number 0; $\leq$ is mapped to the usual less than or equal relation; $\mathbf{S}$ is mapped to successor; and $+$ and $\times$ are mapped to addition and multiplication. In such cases, we have, for example, $[\mathbf{S}(\mathbf{S}(\mathbf{0}) + \mathbf{0}))]^{\mathcal{I}_{\mathbf{N}}} \approx 2$.

As a second example related to logic programming, we mention the family of *Herbrand interpretations* of $L_{\mathbf{N}}$. Each of these shares the same universe and the same mappings for the constant and function symbols. An assignment of a universe, and for the constant and function symbols, is called a *preinterpretation*. In the Herbrand preinterpretation for $L_{\mathbf{N}}$, the universe, denoted $U_{L_{\mathbf{N}}}$, is the set containing $\mathbf{0}$ and all terms that can be constructed from this using the function symbols of the language. This is a little confusing because the terms now play a dual role—as terms constructed from components of the language L, and as elements of the universe $U_{L_{\mathbf{N}}}$. The mapping $\mathcal{C}$ maps the constant symbol $\mathbf{0}$ to $\mathbf{0}$ (considered as an element of $U_{L_{\mathbf{N}}}$). Given a term t in U, the function $\mathcal{F}(\mathbf{S})$ maps t to the term $\mathbf{S}(t)$. Given terms t_1 and t_2, the function $\mathcal{F}(+)$ maps the pair (t_1, t_2) to the term $+(t_1, t_2)$, and the function $\mathcal{F}(\times)$ is defined analogously.

The set of ground atoms of $L_{\mathbf{N}}$ (i.e., the set of atoms that do not contain variables) is sometimes called the *Herbrand base* of $L_{\mathbf{N}}$. There is a natural one-one correspondence

between interpretations of $L_{\mathbf{N}}$ that extend the Herbrand preinterpretation and subsets of the Herbrand base of $L_{\mathbf{N}}$. One Herbrand interpretation of particular interest is the one that mimics the usual interpretation. In particular, this interpretation maps $\leq$ to the set $\{(t_1, t_2) \mid (t_1^{\mathcal{I}_{\mathbf{N}}}, t_2^{\mathcal{I}_{\mathbf{N}}}) \in \leq^{\mathcal{I}_{\mathbf{N}}}\}$.

We now turn to the notion of satisfaction of a formula by an interpretation. The definition is recursive on the structure of formulas; as a result we need the notion of variable assignment to accommodate variables occurring free in formulas. Let L be a language and $\mathcal{I}$ an interpretation of L with universe U. A *variable assignment* for formula φ is a partial function μ : variables of $L \rightarrow U$ whose domain includes all variables free in φ. For terms t, $t^{\mathcal{I},\mu}$ denotes the meaning given to t by $\mathcal{I}$, using μ to interpret the free variables. In addition, if μ is a variable assignment, x is a variable, and $u \in U$, then $\mu[x/u]$ denotes the variable assignment that is identical to μ, except that it maps x to u. We write $I \models \varphi[\mu]$ to indicate that $\mathcal{I}$ *satisfies* φ under μ. This is defined recursively on the structure of formulas in the natural fashion. To indicate the flavor of the definition, we note that $\mathcal{I} \models p(t_1, \ldots, t_n)[\mu]$ if $(t_1^{\mathcal{I},\mu}, \ldots, t_n^{\mathcal{I},\mu}) \in p^{\mathcal{I}}$; $\mathcal{I} \models \exists x \psi[\mu]$ if there is some $u \in U$ such that $\mathcal{I} \models \psi[\mu[x/u]]$; and $\mathcal{I} \models \forall x \psi[\mu]$ if for each $u \in U$, $I \models \psi[\mu[x/u]]$. The Boolean connectives are interpreted in the usual manner. If φ is a sentence, then no variable assignment needs to be specified.

For example, $\mathcal{I}_{\mathbf{N}} \models \forall x \exists y(\neg(x \approx y) \vee x \leq y)$; $\mathcal{I}_{\mathbf{N}} \not\models \mathbf{S}(\mathbf{0}) \leq \mathbf{0}$; and

$$\mathcal{I}_{\mathbf{N}} \models \forall y \forall z(x \approx y \times z \rightarrow (y \approx \mathbf{S}(\mathbf{0}) \vee z \approx \mathbf{S}(\mathbf{0})))[\mu]$$

iff $\mu(x)$ is 1 or a prime number.

An interpretation $\mathcal{I}$ is a *model* of a set Φ of sentences if $\mathcal{I}$ satisfies each formula in Φ. The set Φ is *satisfiable* if it has a model.

Logical implication and equivalence are now defined analogously to the propositional case. Sentence φ logically implies sentence ψ, denoted $\varphi \models \psi$, if each interpretation that satisfies φ also satisfies ψ. There are many straightforward equivalences [e.g., $\neg(\neg\varphi) \equiv \varphi$ and $\neg\forall x\varphi \equiv \exists x \neg\varphi$]. Logical implication is generalized to sets of sentences in the natural manner.

It is known that logical implication, considered as a decision problem, is not recursive. One of the fundamental results of mathematical logic is the development of effective procedures for determining logical equivalence. These are based on the notion of *proofs*, and they provide one way to show that logical implication is r.e. One style of proof, attributed to Hilbert, identifies a family of *inference rules* and a family of *axioms*. An example of an inference rule is *modus ponens*, which states that from formulas φ and $\varphi \rightarrow \psi$ we may conclude ψ. Examples of axioms are all *tautologies* of propositional logic [e.g., $\neg(\varphi \vee \psi) \leftrightarrow (\neg\varphi \wedge \neg\psi)$ for all formulas φ and ψ], and *substitution* (i.e., $\forall x\varphi \rightarrow \varphi_t^x$, where t is an arbitrary term and φ_t^x denotes the formula obtained by simultaneously replacing all occurrences of x free in φ by t). Given a family of inference rules and axioms, a *proof* that set Φ of sentences implies sentence φ is a finite sequence $\psi_0, \psi_1, \ldots, \psi_n = \varphi$, where for each i, either ψ_i is an axiom, or a member of Φ, or it follows from one or more of the previous ψ_j's using an inference rule. In this case we write $\Phi \vdash \varphi$.

The soundness and completeness theorem of Gödel shows that (using *modus ponens* and a specific set of axioms) $\Phi \models \varphi$ iff $\Phi \vdash \varphi$. This important link between $\models$ and $\vdash$ permits the transfer of results obtained in model theory, which focuses primarily on in-

terpretations and models, and proof theory, which focuses primarily on proofs. Notably, a central issue in the study of relational database dependencies (see Part C) has been the search for sound and complete proof systems for subsets of first-order logic that correspond to natural families of constraints.

The model-theoretic and proof-theoretic perspectives lead to two equivalent ways of incorporating equality into first-order languages. Under the model-theoretic approach, the equality predicate $\approx$ is given the meaning $\{(u, u) \mid u \in U\}$ (i.e., normal equality). Under the proof-theoretic approach, a set of *equality axioms* EQ_L is constructed that express the intended meaning of $\approx$. For example, EQ_L includes the sentences $\forall x, y, z(x \approx y \wedge y \approx z \rightarrow x \approx z)$ and $\forall x, y(x \approx y \rightarrow (R(x) \leftrightarrow R(y))$ for each unary predicate symbol R.

Another important result from mathematical logic is the compactness theorem, which can be demonstrated using Gödel's soundness and completeness result. There are two common ways of stating this. The first is that given a (possibly infinite) set of sentences Φ, if $\Phi \models \varphi$ then there is a finite $\Phi' \subseteq \Phi$ such that $\Phi' \models \varphi$. The second is that if each finite subset of Φ is satisfiable, then Φ is satisfiable.

Note that although the compactness theorem guarantees that the Φ in the preceding paragraph has a model, that model is not necessarily finite. Indeed, Φ may only have infinite models. It is of some solace that, among those infinite models, there is surely at least one that is countable (i.e., whose elements can be enumerated: $a_1, a_2, \ldots$). This technically useful result is the Löwenheim-Skolem theorem.

To illustrate the compactness theorem, we show that there is no set Ψ of sentences defining the notion of connectedness in directed graphs. For this we use the language L with two constant symbols, a and b, and one binary relation symbol R, which corresponds to the edges of a directed graph. In addition, because we are working with general first-order logic, both finite and infinite graphs may arise. Suppose now that Ψ is a set of sentences that states that a and b are connected (i.e., that there is a directed path from a to b in R). Let $\Sigma = \{\sigma_i \mid i > 0\}$, where σ_i states "a and b are at least i edges apart from each other." For example, σ_3 might be expressed as

$$\neg R(a, b) \wedge \neg \exists x_1 (R(a, x_1) \wedge R(x_1, b)).$$

It is clear that each finite subset of $\Psi \cup \Sigma$ is satisfiable. By the compactness theorem (second statement), this implies that $\Psi \cup \Sigma$ is satisfiable, so it has a model (say, $\mathcal{I}$). In $\mathcal{I}$, there is no directed path between (the elements of the universe identified by) a and b, and so $\mathcal{I} \not\models \Psi$. This is a contradiction.

Specializations to Database Theory

We close by mentioning the primary differences between the general field of mathematical logic and the specializations made in the study of database theory. The most obvious specialization is that database theory has not generally focused on the use of functions on data values, and as a result it generally omits function symbols from the first-order languages used. The two other fundamental specializations are the focus on finite models and the special use of constant symbols.

An interpretation is *finite* if its universe of discourse is finite. Because most databases

are finite, most of database theory is focused exclusively on finite interpretations. This is closely related to the field of finite model theory in mathematics.

The notion of logical implication for finite interpretations, usually denoted $\models_{\text{fin}}$, is not equivalent to the usual logical implication $\models$. This is most easily seen by considering the compactness theorem. Let $\Phi = \{\sigma_i \mid i > 0\}$, where σ_i states that there are at least i distinct elements in the universe of discourse. Then by compactness, $\Phi \not\models \mathit{false}$, but by the definition of finite interpretation, $\Phi \models_{\text{fin}} \mathit{false}$.

Another way to show that $\models$ and $\models_{\text{fin}}$ are distinct uses computability theory. It is known that $\models$ is r.e. but not recursive, and it is easily seen that $\models_{\text{fin}}$ is co-r.e. Thus if they were equal, $\models$ would be recursive, a contradiction.

The final specialization of database theory concerns assumptions made about the universe of discourse and the use of constant symbols. Indeed, throughout most of this book we use a fixed, countably infinite set of constants, denoted **dom** (for domain elements). Furthermore, the focus is almost exclusively on finite Herbrand interpretations over **dom**. In particular, for distinct constants c and c', all interpretations that are considered satisfy $\neg c \approx c'$.

Most proofs in database theory involving the first-order predicate calculus are based on model theory, primarily because of the emphasis on finite models and because the link between $\models_{\text{fin}}$ and $\vdash$ does not hold. It is thus informative to identify a mechanism for using traditional proof-theoretic techniques within the context of database theory. For this discussion, consider a first-order language with set **dom** of constant symbols and predicate symbols $R_1, \ldots, R_n$. As will be seen in Chapter 3, a database *instance* is a finite Herbrand interpretation $\mathbf{I}$ of this language. Following [Rei84], a family $\Sigma_{\mathbf{I}}$ of sentences is associated with $\mathbf{I}$. This family includes the axioms of equality (mentioned earlier) and

Atoms: $R_i(\vec{a})$ for each $\vec{a}$ in $R_i^{\mathbf{I}}$.

Extension axioms: $\forall \vec{x}(R_i(\vec{x}) \leftrightarrow (\vec{x} \approx \vec{a}_1 \vee \cdots \vee \vec{x} \approx \vec{a}_m))$, where $\vec{a}_1, \ldots, \vec{a}_m$ is a listing of all elements of $R_i^{\mathbf{I}}$, and we are abusing notation by letting $\approx$ range over vectors of terms.

Unique Name axioms: $\neg c \approx c'$ for each distinct pair c, c' of constants occurring in $\mathbf{I}$.

Domain Closure axiom: $\forall x(x \approx c_1 \vee \cdots \vee x \approx c_n)$, where $c_1, \ldots, c_n$ is a listing of all constants occurring in $\mathbf{I}$.

A set of sentences obtained in this manner is termed an *extended relational theory*.

The first two sets of sentences of an extended relational theory express the specific contents of the relations (predicate symbols) of $\mathbf{I}$. Importantly, the Extension sentences ensure that for any (not necessarily Herbrand) interpretation $\mathcal{J}$ satisfying $\Sigma_{\mathbf{I}}$, an n-tuple is in $R_i^{\mathcal{J}}$ iff it equals one of the n-tuples in $R_i^{\mathbf{I}}$. The Unique Name axiom ensures that no pair of distinct constants is mapped to the same element in the universe of $\mathcal{J}$, and the Domain Closure axiom ensures that each element of the universe of $\mathcal{J}$ equals some constant occurring in $\mathbf{I}$. For all intents and purposes, then, any interpretation $\mathcal{J}$ that models $\Sigma_{\mathbf{I}}$ is isomorphic to $\mathbf{I}$, modulo condensing under equivalence classes induced by $\approx^{\mathcal{J}}$. Importantly, the following link with conventional logical implication now holds: For any set Γ of sentences, $\mathbf{I} \models \Gamma$ iff $\Sigma_{\mathbf{I}} \cup \Gamma$ is satisfiable. The perspective obtained through this connection with clas-

sical logic is useful when attempting to extend the conventional relational model (e.g., to incorporate so-called incomplete information, as discussed in Chapter 19).

The Extension axioms correspond to the intuition that a tuple $\vec{a}$ is in relation R only if it is explicitly included in R by the database instance. A more general formulation of this intuition is given by the *closed world assumption* (CWA) [Rei78]. In its most general formulation, the CWA is an inference rule that is used in proof-theoretic contexts. Given a set Σ of sentences describing a (possibly nonconventional) database instance, the CWA states that one can infer a negated atom $R(\vec{a})$ if $\Sigma \nvdash R(\vec{a})$ [i.e., if one cannot prove $R(\vec{a})$ from Σ using conventional first-order logic]. In the case where Σ is an extended relational theory this gives no added information, but in other contexts (such as deductive databases) it does. The CWA is related in spirit to the *negation as failure* rule of [Cla78].

the semantics of nr-datalog programs. An *nr-datalog¬ query* is a query defined by some nr-datalog¬ 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

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

Query (5.3) is answered by

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

Care must be taken when forming nr-datalog¬ 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.)

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

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¬ program with equality can be simulated by an nr-datalog¬ 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¬ 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 $\mathbf{R}$, the *base formulas* include, as before, atoms over $\mathbf{R}$ 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 $\mathbf{R}$ include the base formulas and formulas of the form

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

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

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

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

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

As with conjunctive calculus,

$\exists x_1, x_2, \ldots, x_m \varphi$ abbreviates $\exists x_1 \exists x_2 \ldots \exists x_m \varphi$, and
$\forall x_1, x_2, \ldots, x_m \varphi$ 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 *free*(φ) 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

$\{\langle e_1, \ldots, e_m \rangle : A_1, \ldots, A_m \mid \varphi\}$, in the named perspective,
$\{e_1, \ldots, e_m \mid \varphi\}$, in the unnamed perspective,
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 *free*(φ).

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:

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

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 $\mathbf{R}$ is a pair $(\mathbf{d}, \mathbf{I})$, where $\mathbf{I}$ is an instance over $\mathbf{R}$ and $adom(\mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$. A calculus formula φ is *interpretable* over $(\mathbf{d},\mathbf{I})$ if $adom(\varphi) \subseteq \mathbf{d}$. In this case, if ν is a valuation over $free(\varphi)$ with range contained in $\mathbf{d}$, then $\mathbf{I}$ *satisfies* φ for ν *relative* to $\mathbf{d}$, 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 $\mathbf{d}$ 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 $\mathbf{I}$, 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 $\mathbf{R}$ be a schema, $q = \{e_1, \ldots, e_n \mid \varphi\}$ a calculus query over $\mathbf{R}$, and $(\mathbf{d}, \mathbf{I})$ a relativized instance over $\mathbf{R}$. Then

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

the *image* of $\mathbf{I}$ under q *relative to* $\mathbf{d}$ 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 $\mathbf{d}$ 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 $\mathbf{I}$ relative to $\mathbf{d}$ 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

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

This illustrates that under an interpretation relative to a set $\mathbf{d}$, a calculus query q on input $\mathbf{I}$ 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 $\mathbf{I}$ are independent of whether $\mathbf{dom}$ or some subset $\mathbf{d}$ 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 $\mathbf{I}$, the *natural* (or *unrestricted*) interpretation of q on $\mathbf{I}$, 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 $\text{CALC}_{di} \sqsubseteq \text{CALC}_{adom}$.

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 $\sqsubseteq \text{CALC}_{sr}$. 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	$\exists x\varphi$	$\leftrightarrow$	$\exists y\varphi_y^x$ (y not free in φ)
17	$\forall x\varphi$	$\leftrightarrow$	$\forall y\varphi_y^x$ (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 $\mathbf{R}$ are *equivalent*, denoted $\varphi \equiv \psi$, if for each $\mathbf{I}$ over $\mathbf{R}$, $\mathbf{d} \subseteq \mathbf{dom}$, and valuation ν with range $\subseteq \mathbf{d}$

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

(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 $\texttt{SRNF}(\varphi)$. A formula φ (query $\{\vec{e} \mid \varphi\}$) is in SRNF if $\texttt{SRNF}(\varphi) = \varphi$.

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

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

Transforming the query of Example 5.3.2 into SRNF yields

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

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 φ **of**

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

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 $rr(\mathtt{SRNF}(\varphi)) = free(\varphi)$. 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 $\sqsubseteq \mathrm{CALC}_{sr}$.

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

The proof of this theorem involves several steps. As seen earlier, the algebra $\sqsubseteq \text{CALC}_{sr}$. 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 $rr(\psi) = rr(\psi_1) = \cdots = rr(\psi_n) = free(\psi)$;
(ii) $\psi = \exists \vec{x} \psi_1$ and $rr(\psi) = free(\psi_1)$; or
(iii) $\psi = \neg \psi_1$ and $rr(\psi) = free(\psi_1)$.

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 $rr(\psi) = free(\psi)$, but $rr(\xi_1 \vee \cdots \vee \xi_m) \neq free(\xi_1 \vee \cdots \vee \xi_m)$. 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.