is the head relation defined by this rule. More generally, had tuple $(1,2)$ appeared n times in R and tuple $(2,3)$ appeared m times in S, then tuple $(1,3)$ would appear nm times in H. □

If a relation is defined by several rules, then the result is the bag-union of whatever tuples are produced by each rule.

Example 5.23: Consider a relation H defined by the two rules

```
H(x,y) ← S(x,y) AND x>1
H(x,y) ← S(x,y) AND y<5
```

where relation $S(B,C)$ is as in Example 5.22; that is, $S = \{(2,3),\ (4,5),\ (4,5)\}$. The first rule puts each of the three tuples of S into H, since they each have a first component greater than 1. The second rule puts only the tuple $(2,3)$ into H, since $(4,5)$ does not satisfy the condition $y < 5$. Thus, the resulting relation H has two copies of the tuple $(2,3)$ and two copies of the tuple $(4,5)$. □

5.3.7 Exercises for Section 5.3

Exercise 5.3.1: Write each of the queries of Exercise 2.4.1 in Datalog. You should use only safe rules, but you may wish to use several IDB predicates corresponding to subexpressions of complicated relational-algebra expressions.

Exercise 5.3.2: Write each of the queries of Exercise 2.4.3 in Datalog. Again, use only safe rules, but you may use several IDB predicates if you like.

!! **Exercise 5.3.3:** The requirement we gave for safety of Datalog rules is sufficient to guarantee that the head predicate has a finite relation if the predicates of the relational subgoals have finite relations. However, this requirement is too strong. Give an example of a Datalog rule that violates the condition, yet whatever finite relations we assign to the relational predicates, the head relation will be finite.

5.4 Relational Algebra and Datalog

Each of the relational-algebra operators of Section 2.4 can be mimicked by one or several Datalog rules. In this section we shall consider each operator in turn. We shall then consider how to combine Datalog rules to mimic complex algebraic expressions. It is also true that any single safe Datalog rule can be expressed in relational algebra, although we shall not prove that fact here. However, Datalog queries are more powerful than relational algebra when several rules are allowed to interact; they can express recursions that are not expressable in the algebra (see Example 5.35).

5.4.1 Boolean Operations

The boolean operations of relational algebra — union, intersection, and set difference — can each be expressed simply in Datalog. Here are the three techniques needed. We assume R and S are relations with the same number of attributes, n. We shall describe the needed rules using `Answer` as the name of the head predicate in all cases. However, we can use anything we wish for the name of the result, and in fact it is important to choose different predicates for the results of different operations.

- To take the union $R \cup S$, use two rules and n distinct variables

  $$a_1, a_2, \ldots, a_n$$

  One rule has $R(a_1, a_2, \ldots, a_n)$ as the lone subgoal and the other has $S(a_1, a_2, \ldots, a_n)$ alone. Both rules have the head $Answer(a_1, a_2, \ldots, a_n)$. As a result, each tuple from R and each tuple of S is put into the answer relation.

- To take the intersection $R \cap S$, use a rule with body

  $$R(a_1, a_2, \ldots, a_n) \text{ AND } S(a_1, a_2, \ldots, a_n)$$

  and head $Answer(a_1, a_2, \ldots, a_n)$. Then, a tuple is in the answer relation if and only if it is in both R and S.

- To take the difference $R - S$, use a rule with body

  $$R(a_1, a_2, \ldots, a_n) \text{ AND NOT } S(a_1, a_2, \ldots, a_n)$$

  and head $Answer(a_1, a_2, \ldots, a_n)$. Then, a tuple is in the answer relation if and only if it is in R but not in S.

Example 5.24: Let the schemas for the two relations be $R(A, B, C)$ and $S(A, B, C)$. To avoid confusion, we use different predicates for the various results, rather than calling them all `Answer`.

To take the union $R \cup S$ we use the two rules:

```
1. U(x,y,z) ← R(x,y,z)
2. U(x,y,z) ← S(x,y,z)
```

Rule (1) says that every tuple in R is a tuple in the IDB relation U. Rule (2) similarly says that every tuple in S is in U.

To compute $R \cap S$, we use the rule

```
I(a,b,c) ← R(a,b,c) AND S(a,b,c)
```

Finally, the rule

```
D(a,b,c) ← R(a,b,c) AND NOT S(a,b,c)
```

computes the difference $R - S$. □

Variables Are Local to a Rule

Notice that the names we choose for variables in a rule are arbitrary and have no connection to the variables used in any other rule. The reason there is no connection is that each rule is evaluated alone and contributes tuples to its head's relation independent of other rules. Thus, for instance, we could replace the second rule of Example 5.24 by

```
U(a,b,c) ← S(a,b,c)
```

while leaving the first rule unchanged, and the two rules would still compute the union of R and S. Note, however, that when substituting one variable u for another variable v within a rule, we must substitute u for all occurrences of v within the rule. Moreover, the substituting variable u that we choose must not be a variable that already appears in the rule.

5.4.2 Projection

To compute a projection of a relation R, we use one rule with a single subgoal with predicate R. The arguments of this subgoal are distinct variables, one for each attribute of the relation. The head has an atom with arguments that are the variables corresponding to the attributes in the projection list, in the desired order.

Example 5.25: Suppose we want to project the relation

```
Movies(title, year, length, genre, studioName, producerC#)
```

onto its first three attributes — `title`, `year`, and `length`. The rule

```
P(t,y,l) ← Movies(t,y,l,g,s,p)
```

serves, defining a relation called P to be the result of the projection. □

5.4.3 Selection

Selections can be somewhat more difficult to express in Datalog. The simple case is when the selection condition is the `AND` of one or more arithmetic comparisons. In that case, we create a rule with

1. One relational subgoal for the relation upon which we are performing the selection. This atom has distinct variables for each component, one for each attribute of the relation.

2. For each comparison in the selection condition, an arithmetic subgoal that is identical to this comparison. However, while in the selection condition an attribute name was used, in the arithmetic subgoal we use the corresponding variable, following the correspondence established by the relational subgoal.

Example 5.26 : The selection

$$\sigma_{length \geq 100 \text{ AND } studioName='\text{Fox}'}(\text{Movies})$$

can be written as a Datalog rule

```
S(t,y,l,g,s,p) ← Movies(t,y,l,g,s,p) AND l ≥ 100 AND s = 'Fox'
```

The result is the relation S. Note that l and s are the variables corresponding to attributes `length` and `studioName` in the standard order we have used for the attributes of `Movies`. □

Now, let us consider selections that involve the `OR` of conditions. We cannot necessarily replace such selections by single Datalog rules. However, selection for the `OR` of two conditions is equivalent to selecting for each condition separately and then taking the union of the results. Thus, the `OR` of n conditions can be expressed by n rules, each of which defines the same head predicate. The ith rule performs the selection for the ith of the n conditions.

Example 5.27 : Let us modify the selection of Example 5.26 by replacing the `AND` by an `OR` to get the selection:

$$\sigma_{length \geq 100 \text{ OR } studioName='\text{Fox}'}(\text{Movies})$$

That is, find all those movies that are either long or by Fox. We can write two rules, one for each of the two conditions:

```
1. S(t,y,l,g,s,p) ← Movies(t,y,l,g,s,p) AND l ≥ 100
2. S(t,y,l,g,s,p) ← Movies(t,y,l,g,s,p) AND s = 'Fox'
```

Rule (1) produces movies at least 100 minutes long, and rule (2) produces movies by Fox. □

Even more complex selection conditions can be formed by several applications, in any order, of the logical operators `AND`, `OR`, and `NOT`. However, there is a widely known technique, which we shall not present here, for rearranging any such logical expression into "disjunctive normal form," where the expression is the disjunction (`OR`) of "conjuncts." A *conjunct*, in turn, is the `AND` of "literals," and a *literal* is either a comparison or a negated comparison.[2]

[2]See, e.g., A. V. Aho and J. D. Ullman, *Foundations of Computer Science*, Computer Science Press, New York, 1992.

We can represent any literal by a subgoal, perhaps with a NOT in front of it. If the subgoal is arithmetic, the NOT can be incorporated into the comparison operator. For example, NOT x $\geq$ 100 can be written as x < 100. Then, any conjunct can be represented by a single Datalog rule, with one subgoal for each comparison. Finally, every disjunctive-normal-form expression can be written by several Datalog rules, one rule for each conjunct. These rules take the union, or OR, of the results from each of the conjuncts.

Example 5.28 : We gave a simple instance of this algorithm in Example 5.27. A more difficult example can be formed by negating the condition of that example. We then have the expression:

$$\sigma_{\texttt{NOT}\ (length \geq 100\ \texttt{OR}\ studioName = \texttt{'Fox'})}(\texttt{Movies})$$

That is, find all those movies that are neither long nor by Fox.

Here, a NOT is applied to an expression that is itself not a simple comparison. Thus, we must push the NOT down the expression, using one form of *DeMorgan's laws*, which says that the negation of an OR is the AND of the negations. That is, the selection can be rewritten:

$$\sigma_{(\texttt{NOT}\ (length \geq 100))\ \texttt{AND}\ (\texttt{NOT}\ (studioName = \texttt{'Fox'}))}(\texttt{Movies})$$

Now, we can take the NOT's inside the comparisons to get the expression:

$$\sigma_{length < 100\ \texttt{AND}\ studioName \neq \texttt{'Fox'}}(\texttt{Movies})$$

This expression can be converted into the Datalog rule

S(t,y,l,g,s,p) $\leftarrow$ Movies(t,y,l,g,s,p) AND l $<$ 100 AND s $\neq$ 'Fox'

□

Example 5.29 : Let us consider a similar example where we have the negation of an AND in the selection. Now, we use the second form of DeMorgan's law, which says that the negation of an AND is the OR of the negations. We begin with the algebraic expression

$$\sigma_{\texttt{NOT}\ (length \geq 100\ \texttt{AND}\ studioName = \texttt{'Fox'})}(\texttt{Movies})$$

That is, find all those movies that are not both long and by Fox.

We apply DeMorgan's law to push the NOT below the AND, to get:

$$\sigma_{(\texttt{NOT}\ (length \geq 100))\ \texttt{OR}\ (\texttt{NOT}\ (studioName = \texttt{'Fox'}))}(\texttt{Movies})$$

Again we take the NOT's inside the comparisons to get:

$$\sigma_{length < 100\ \texttt{OR}\ studioName \neq \texttt{'Fox'}}(\texttt{Movies})$$

Finally, we write two rules, one for each part of the OR. The resulting Datalog rules are:

1. S(t,y,l,g,s,p) $\leftarrow$ Movies(t,y,l,g,s,p) AND l $<$ 100
2. S(t,y,l,g,s,p) $\leftarrow$ Movies(t,y,l,g,s,p) AND s $\neq$ 'Fox'

□

5.4.4 Product

The product of two relations $R \times S$ can be expressed by a single Datalog rule. This rule has two subgoals, one for R and one for S. Each of these subgoals has distinct variables, one for each attribute of R or S. The IDB predicate in the head has as arguments all the variables that appear in either subgoal, with the variables appearing in the R-subgoal listed before those of the S-subgoal.

Example 5.30: Let us consider the two three-attribute relations R and S from Example 5.24. The rule

```
P(a,b,c,x,y,z) ← R(a,b,c) AND S(x,y,z)
```

defines P to be $R \times S$. We have arbitrarily used variables at the beginning of the alphabet for the arguments of R and variables at the end of the alphabet for S. These variables all appear in the rule head. □

5.4.5 Joins

We can take the natural join of two relations by a Datalog rule that looks much like the rule for a product. The difference is that if we want $R \bowtie S$, then we must use the same variable for attributes of R and S that have the same name and must use different variables otherwise. For instance, we can use the attribute names themselves as the variables. The head is an IDB predicate that has each variable appearing once.

Example 5.31: Consider relations with schemas $R(A,B)$ and $S(B,C,D)$. Their natural join may be defined by the rule

```
J(a,b,c,d) ← R(a,b) AND S(b,c,d)
```

Notice how the variables used in the subgoals correspond in an obvious way to the attributes of the relations R and S. □

We also can convert theta-joins to Datalog. Recall from Section 2.4.12 how a theta-join can be expressed as a product followed by a selection. If the selection condition is a conjunct, that is, the AND of comparisons, then we may simply start with the Datalog rule for the product and add additional, arithmetic subgoals, one for each of the comparisons.

Example 5.32: Consider the relations $U(A,B,C)$ and $V(B,C,D)$ and the theta-join:

$$U \bowtie_{A<D \texttt{ AND } U.B \neq V.B} V$$

We can construct the Datalog rule

```
J(a,ub,uc,vb,vc,d)  ←  U(a,ub,uc) AND V(vb,vc,d) AND
                       a < d AND ub ≠ vb
```

to perform the same operation. We have used *ub* as the variable corresponding to attribute B of U, and similarly used *vb*, *uc*, and *vc*, although any six distinct variables for the six attributes of the two relations would be fine. The first two subgoals introduce the two relations, and the second two subgoals enforce the two comparisons that appear in the condition of the theta-join. □

If the condition of the theta-join is not a conjunction, then we convert it to disjunctive normal form, as discussed in Section 5.4.3. We then create one rule for each conjunct. In this rule, we begin with the subgoals for the product and then add subgoals for each literal in the conjunct. The heads of all the rules are identical and have one argument for each attribute of the two relations being theta-joined.

Example 5.33: In this example, we shall make a simple modification to the algebraic expression of Example 5.32. The `AND` will be replaced by an `OR`. There are no negations in this expression, so it is already in disjunctive normal form. There are two conjuncts, each with a single literal. The expression is:

$$U \bowtie_{A<D \texttt{ OR } U.B \neq V.B} V$$

Using the same variable-naming scheme as in Example 5.32, we obtain the two rules

```
1. J(a,ub,uc,vb,vc,d) ← U(a,ub,uc) AND V(vb,vc,d) AND a < d
2. J(a,ub,uc,vb,vc,d) ← U(a,ub,uc) AND V(vb,vc,d) AND ub ≠ vb
```

Each rule has subgoals for the two relations involved plus a subgoal for one of the two conditions $A < D$ or $U.B \neq V.B$. □

5.4.6 Simulating Multiple Operations with Datalog

Datalog rules are not only capable of mimicking a single operation of relational algebra. We can in fact mimic any algebraic expression. The trick is to look at the expression tree for the relational-algebra expression and create one IDB predicate for each interior node of the tree. The rule or rules for each IDB predicate is whatever we need to apply the operator at the corresponding node of the tree. Those operands of the tree that are extensional (i.e., they are relations of the database) are represented by the corresponding predicate. Operands that are themselves interior nodes are represented by the corresponding IDB predicate. The result of the algebraic expression is the relation for the predicate associated with the root of the expression tree.

Example 5.34: Consider the algebraic expression

$$\pi_{title,year}\Big(\sigma_{length\geq 100}(\texttt{Movies}) \cap \sigma_{studioName=\texttt{'Fox'}}(\texttt{Movies})\Big)$$

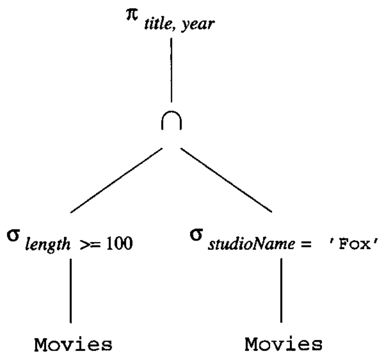

Figure 5.9: Expression tree

```
1. W(t,y,l,g,s,p) ← Movies(t,y,l,g,s,p) AND l ≥ 100
2. X(t,y,l,g,s,p) ← Movies(t,y,l,g,s,p) AND s = 'Fox'
3. Y(t,y,l,g,s,p) ← W(t,y,l,g,s,p) AND X(t,y,l,g,s,p)
4. Answer(t,y) ← Y(t,y,l,g,s,p)
```

Figure 5.10: Datalog rules to perform several algebraic operations

from Example 2.17, whose expression tree appeared in Fig. 2.18. We repeat this tree as Fig. 5.9. There are four interior nodes, so we need to create four IDB predicates. Each of these predicates has a single Datalog rule, and we summarize all the rules in Fig. 5.10.

The lowest two interior nodes perform simple selections on the EDB relation `Movies`, so we can create the IDB predicates W and X to represent these selections. Rules (1) and (2) of Fig. 5.10 describe these selections. For example, rule (1) defines W to be those tuples of `Movies` that have a length at least 100.

Then rule (3) defines predicate Y to be the intersection of W and X, using the form of rule we learned for an intersection in Section 5.4.1. Finally, rule (4) defines the answer to be the projection of Y onto the `title` and `year` attributes. We here use the technique for simulating a projection that we learned in Section 5.4.2.

Note that, because Y is defined by a single rule, we can substitute for the Y subgoal in rule (4) of Fig. 5.10, replacing it with the body of rule (3). Then, we can substitute for the W and X subgoals, using the bodies of rules (1) and (2). Since the `Movies` subgoal appears in both of these bodies, we can eliminate one copy. As a result, the single rule

```
Answer(t,y) ← Movies(t,y,l,g,s,p) AND l ≥ 100 AND s = 'Fox'
```

suffices. □

5.4.7 Comparison Between Datalog and Relational Algebra

We see from Section 5.4.6 that every expression in the basic relational algebra of Section 2.4 can be expressed as a Datalog query. There are operations in the extended relational algebra, such as grouping and aggregation from Section 5.2, that have no corresponding features in the Datalog version we have presented here. Likewise, Datalog does not support bag operations such as duplicate elimination.

It is also true that any single Datalog rule can be expressed in relational algebra. That is, we can write a query in the basic relational algebra that produces the same set of tuples as the head of that rule produces.

However, when we consider collections of Datalog rules, the situation changes. Datalog rules can express recursion, which relational algebra can not. The reason is that IDB predicates can also be used in the bodies of rules, and the tuples we discover for the heads of rules can thus feed back to rule bodies and help produce more tuples for the heads. We shall not discuss here any of the complexities that arise, especially when the rules have negated subgoals. However, the following example will illustrate recursive Datalog.

Example 5.35: Suppose we have a relation `Edge(X,Y)` that says there is a directed edge (arc) from node X to node Y. We can express the transitive closure of the edge relation, that is, the relation `Path(X,Y)` meaning that there is a path of length 1 or more from node X to node Y, as follows:

```
1. Path(X,Y) ← Edge(X,Y)
2. Path(X,Y) ← Edge(X,Z) AND Path(Z,Y)
```

Rule (1) says that every edge is a path. Rule (2) says that if there is an edge from node X to some node Z and a path from Z to Y, then there is also a path from X to Y. If we apply Rule (1) and then Rule (2), we get the paths of length 2. If we take the *Path* facts we get from this application and use them in another application of Rule (2), we get paths of length 3. Feeding those *Path* facts back again gives us paths of length 4, and so on. Eventually, we discover all possible path facts, and on one round we get no new facts. At that point, we can stop. If we haven't discovered the fact $Path(a, b)$, then there really is no path in the graph from node a to node b. □

5.4.8 Exercises for Section 5.4

Exercise 5.4.1: Let $R(a, b, c)$, $S(a, b, c)$, and $T(a, b, c)$ be three relations. Write one or more Datalog rules that define the result of each of the following expressions of relational algebra:

a) $R \cup S$.

b) $R \cap S$.

c) $R - S$.

d) $(R \cup S) - T$.

! e) $(R - S) \cap (R - T)$.

f) $\pi_{a,b}(R)$.

! g) $\pi_{a,b}(R) \cap \rho_{U(a,b)}\big(\pi_{b,c}(S)\big)$.

Exercise 5.4.2: Let $R(x, y, z)$ be a relation. Write one or more Datalog rules that define $\sigma_C(R)$, where C stands for each of the following conditions:

a) $x = y$.

b) $x < y$ AND $y < z$.

c) $x < y$ OR $y < z$.

d) NOT $(x < y$ OR $x > y)$.

! e) NOT $\big((x < y$ OR $x > y)$ AND $y < z\big)$.

! f) NOT $\big((x < y$ OR $x < z)$ AND $y < z\big)$.

Exercise 5.4.3: Let $R(a, b, c)$, $S(b, c, d)$, and $T(d, e)$ be three relations. Write single Datalog rules for each of the natural joins:

a) $R \bowtie S$.

b) $S \bowtie T$.

c) $(R \bowtie S) \bowtie T$. (*Note*: since the natural join is associative and commutative, the order of the join of these three relations is irrelevant.)

Exercise 5.4.4: Let $R(x, y, z)$ and $S(x, y, z)$ be two relations. Write one or more Datalog rules to define each of the theta-joins $R \bowtie_C S$, where C is one of the conditions of Exercise 5.4.2. For each of these conditions, interpret each arithmetic comparison as comparing an attribute of R on the left with an attribute of S on the right. For instance, $x < y$ stands for $R.x < S.y$.

! **Exercise 5.4.5:** It is also possible to convert Datalog rules into equivalent relational-algebra expressions. While we have not discussed the method of doing so in general, it is possible to work out many simple examples. For each of the Datalog rules below, write an expression of relational algebra that defines the same relation as the head of the rule.

a) `P(x,y) ← Q(x,z) AND R(z,y)`

b) `P(x,y) ← Q(x,z) AND Q(z,y)`

c) `P(x,y) ← Q(x,z) AND R(z,y) AND x < y`